Datenbanksysteme II

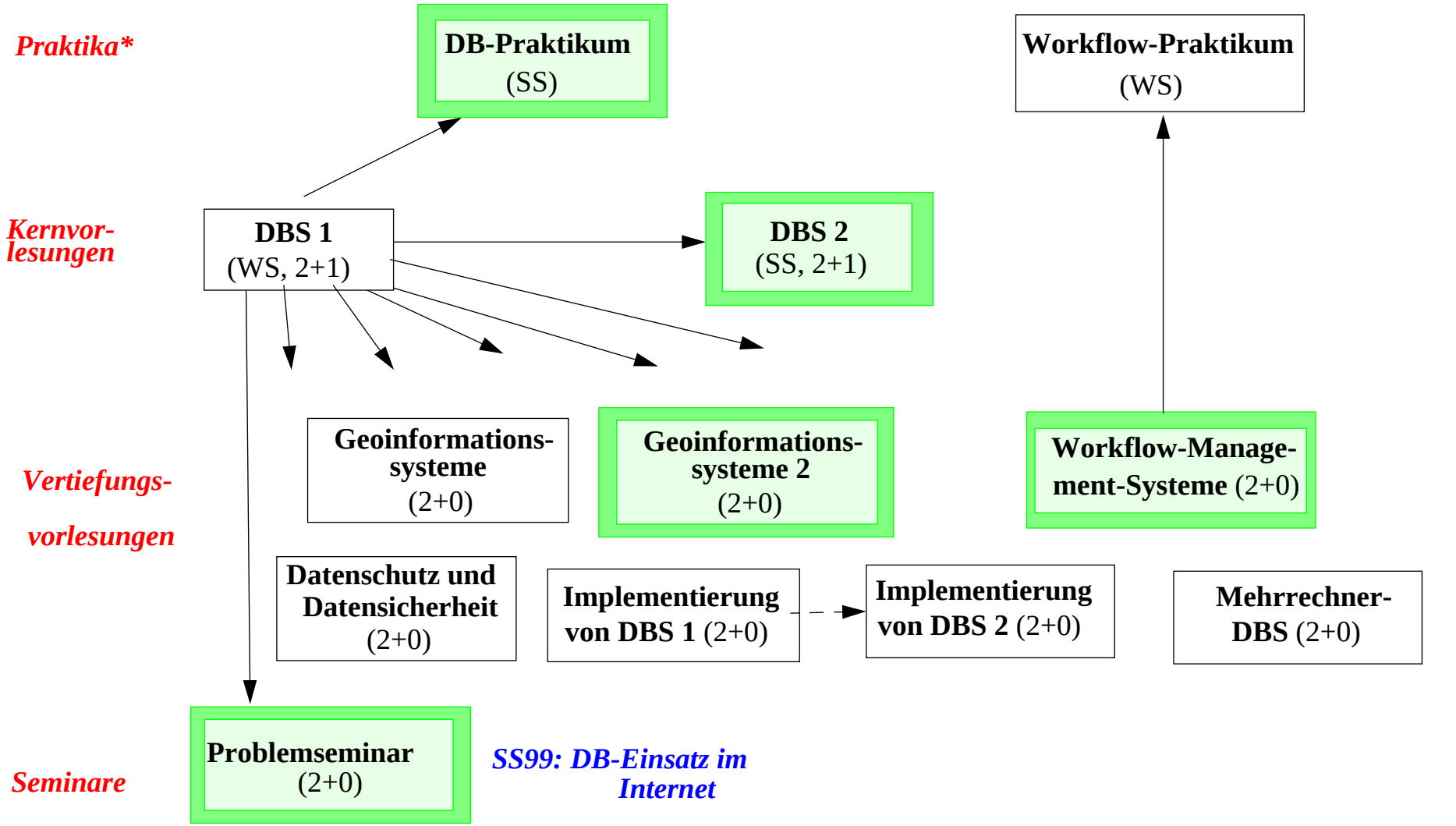
Prof. Dr. E. Rahm

Sommersemester 1999

Universität Leipzig

Institut für Informatik

Lehrveranstaltungen zu "Datenbanken"(SS99)



* *Wertung als 4-stündige Vertiefung*

Vorläufiges Inhaltsverzeichnis

1. Klassen und Einsatzfelder von DBS

- Anforderungen neuartiger DB-Anwendungen (CAD etc.)
- Beschränkungen des Relationemodells
- Datenmodelle: Semantische DM, Objektorientierte DBS, Objektrelationale DBS, Deduktive DBS
- Anwendungsfelder: Information-Retrieval, Multimedia-DBS, GIS, Decision Support / Data Warehouses

2. Grundkonzepte von objektorientierten DBS

- Grundlagen und Konzepte
- Struktureigenschaften
- Objektorientierte Verarbeitung

3. ODMG-Standard

- Kooperation in heterogenen Umgebungen (CORBA-Standard)
- Objektmodell
- Objektdefinition (ODL)
- Anfragesprache (OQL)

4. Beispielrealisierungen von OODBS

- NF^2
- GemStone
- O_2

Vorläufiges Inhaltsverzeichnis (2)

5. Architektur von OODBS

- Workstation/Server-Kooperation
- Page-Server, Query-Server etc.
- Unterstützung langer Entwurfsvorgänge

6. Objektrelationale DBS / SQL3

- Abgrenzung
- SQL3-Überblick
- Abstrakte Datentypen
- Typ- und Tabellenhierarchien

7. Heterogene Datenbanken

- Autonomie und Heterogenität
- Verteilte Transaktionssysteme
- Föderative Mehrrechner-DBS
- Standards, Lösungsansätze: Remote Database Access (RDA), X/OPEN DTP, OLE/DB, ...

8. Data Warehouses

- Architektur, Aufgaben
- Modellierung: ROLAP/MOLAP, Star-Schema, ...
- Operationen: Drill Down, Roll Up, Cube-Operator, ...)

Literatur

- Heuer, A.: *Objektorientierte Datenbanksysteme.* Addison-Wesley, 2. Auflage, 1997
- Lausen, G.; Vossen, G.: *Objektorientierte Datenbanken*. Oldenbourg, 1996
- Saake, G.; Türker, C.; Schmitt, I.: *Objektdatenbanken.* Thomson 1997
- Cattel, R.G.G. et al.: *Object Database Standard 2.0.* Addison Wesley, 1997
- Stonebraker, M.: *Object-Relational DBMS - The Next Great Wave*. Morgan Kaufmann, 1996
- Khosafian, S., Baker, A. B.: *MultiMedia and Imaging Databases.* Morgan Kaufmann, 1996
- Rahm, E.: *Mehrrechner-Datenbanksysteme.* Addison-Wesley, 1994
- Anahory, S.; Murray, D.: *Data Warehouse - Planung, Implementierung und Administration,* Addison-Wesley, 1997

DBS-Einsatz im administrativ-betriebswirtschaftlichen Bereich

- Einsatz in “kommerziellen” Anwendungen
  - Produktionsplanung und -steuerung
  - Lagerverwaltung
  - Personalverwaltung ...
- als Kern von Transaktionssystemen für die interaktive
  - Auskunftsbearbeitung
  - Buchung
  - Datenerfassung
- Spezialsysteme auf den Hochleistungsbereich zugeschnitten
  - einfache und kurze Transaktionen
  - sehr hohe Transaktionsraten (> 1000 TPS)
- meist einfache Verhältnisse
  - überschaubare Anzahl von Relationen
  - nur wenige, einfache Datentypen
  - einfache Operationen
  - einfache Transaktionen

Anspruchsvolle DB-Anwendungen

- Anwendungsklassen
  - Ingenieur-Anwendungen (CAD/CAM)
  - VLSI-Entwurf
  - Geographische Informationssysteme (GIS)
  - Expertensysteme (XPS)
  - Büro-Anwendungen,
  - multimediale Anwendungen
    - • •
- Aufgaben/Eigenschaften
  - Verwaltung und Handhabung von komplexen Objekten
  - stärkere Objektorientierung
  - Gewährleistung komplexer Integritätsbedingungen
  - Unterstützung von Zeit/Versionen
  - Schnittstelle zugeschnitten auf Anwendungsklasse
- große Bandbreite unterschiedlicher Anwendungen mit unterschiedlichsten Anforderungen

Objekt-Darstellung

- kommerzielle Anwendung:
  pro Objekt gibt es genau eine Satzausprägung, die alle beschreibenden Attribute enthält

Schema

ANGESTELLTER

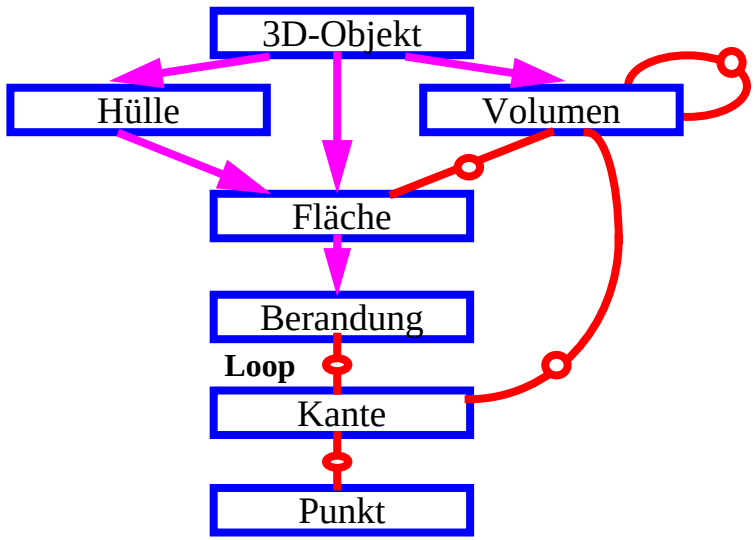
Satztyp (Relation)

Ausprägungen

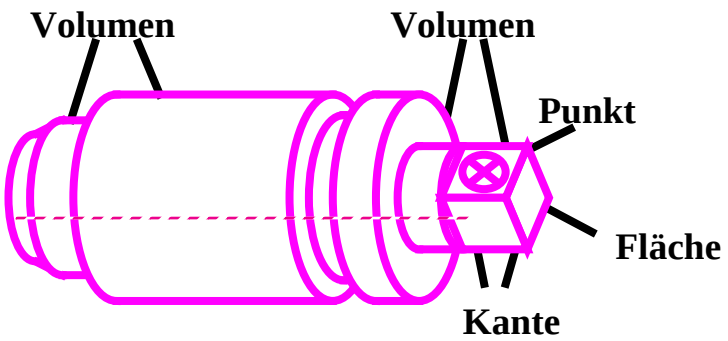
PNR	NAME	TAETIGKEIT	GEHALT	ALTER
496	Peinl	Pfoertner	2100	63
497	Kinzinger	Kopist	2800	25
498	Meyweg	Kalligraph	4500	56

- CAD-Anwendung: das komplexe Objekt „Werkstück“ setzt sich aus einfacheren (komplexen) Objekten verschiedenen Typs zusammen.

Schema



Objektausprägung



Beispiel: Modellierung von Polyedern

ERM

Modellierung im Relationenmodell

```
CREATE TABLE Polyeder
  (polyid     :  INTEGER,
  anzflächen:  INTEGER,
  PRIMARY KEY (polyid));
```

```
CREATE TABLE Fläche
  (fid          :  INTEGER,
  anzkanten :  INTEGER,
  pref          :  INTEGER,
  PRIMARY KEY (fid),
  FOREIGN KEY (pref)
REFERENCES Polyeder);
```

```
CREATE TABLE Kante
  (kid          :  INTEGER,
  ktyp          :  CHAR(5),
  PRIMARY KEY (kid));
```

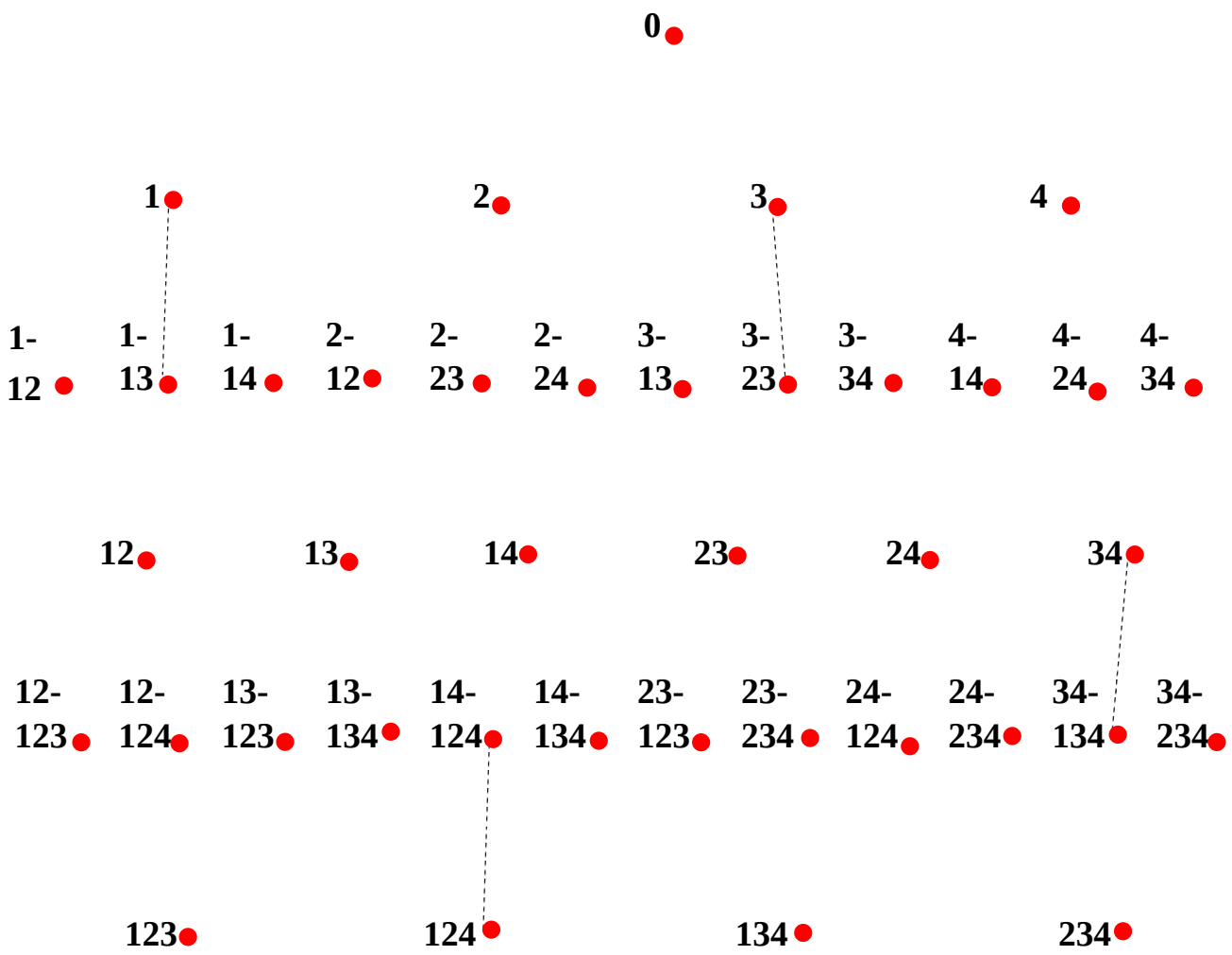
```
CREATE TABLE Punkt
  (pid          :  INTEGER,
  x, y, z       :  INTEGER,
  PRIMARY KEY (pid));
```

```
CREATE TABLE FK_Rel
  (fid    :     INTEGER,
  kid     :     INTEGER,
  PRIMARY KEY  (fid, kid),
  FOREIGN KEY (fid)
          REFERENCES Fläche,
  FOREIGN KEY (kid)
          REFERENCES Kante);
```

```
CREATE TABLE KP_Rel
  (kid  :       INTEGER,
  pid   :       INTEGER,
  PRIMARY KEY  (kid, pid),
  FOREIGN KEY (kid)
        REFERENCES Kante,
  FOREIGN KEY (pid)
        REFERENCES Punkt);
```

Relationenmodell - angemessene Modellierung?

Darstellung eines Tetraeder mit vid = 0



Relationen

Polyeder

Fläche

FK-Rel

Kante

KP-Rel

Punkt

Referenzbeispiel in relationaler Darstellung: Anfragen

- Finde alle Punkte, die zu Flächenobjekten mit F.fid < 3 gehören

```
SELECT   F.fid, P.x, P.y, P.z
FROM     Punkt P, KP-Rel S, Kante K, FK-Rel T, Fläche F
WHERE    F.fid < 3
   AND   P.pid = S.pid                  /*  Rekonstruktion */
   AND   S.kid  = K.kid                 /*  komplexer Objekte*/
   AND   K.kid  = T.kid                 /*  zur Laufzeit   */
   AND   T.fid  =  F.fid;
```

- Symmetrischer Zugriff: Finde alle Flächen, die mit Punkt (50,44,75) assoziiert sind

```
SELECT   F.fid
FROM     Punkt P, KP-Rel S, Kante K, FK-Rel T, Fläche F
WHERE    P.x = 50 AND P.y = 44 AND P.z = 75
   AND   P.pid = S.pid
   AND   S.kid  = K.kid
   AND   K.kid  = T.kid
   AND   T.fid  =  F.fid;
```

Beschränkungen des Relationenmodells

- Datenmodellierung
  - nur einfach strukturierte Datenobjekte (satzorientiert, festes Format)
  - nur einfache (Standard-) Datentypen
  - nur einfache Integritätsbedingungen
  - keine Unterstützung der Abstraktionskonzepte (Generalisierung, Aggregation)
  - keine Unterstützung von Versionen und Zeit
  - keine Spezifikation von "Verhalten" (Funktionen)

- begrenzte Auswahlmächtigkeit der Fragesprachen
  - nicht sämtliche Berechnungen möglich
  - keine Unterstützung von Rekursion (Berechnung der transitiven Hülle)
  - Änderungen jeweils auf 1 Relation beschränkt, ...

- umständliche Einbettung in Programmiersprachen (impedance mismatch)

- auf kurze Transaktionen zugeschnitten (ACID)

- schlechte Effizienz für anspruchsvolle Anwendungen

Semantische Datenmodelle

- Forderungen an geeignetes Datenmodell
  - Definition von
    - statischen Eigenschaften (Objekte, Attribute, Beziehungen),
    - dynamischen Eigenschaften (Operationen) sowie
    - statischen und dynamischen Integritätsbedingungen
  - direkte Modellierbarkeit von Anwendungsobjekten (=> Unterstützung komplexer Objekte)
  - Mächtigkeit: Unterstützung spezieller semantischer Konstrukte (n:m-Beziehungen, etc.)
  - Abstraktionsvermögen (Information hiding)
  - Exakte, formale Definition zur Gewährleistung von Konsistenz (Nicht-Widersprüchlichkeit), Vollständigkeit und geringer Redundanz

- Motivation für semantische DM: konventionelle Datenmodelle (z.B. RM) besitzen nur geringe Kenntnisse über die Bedeutung (Semantik) der Daten
  - unzureichende Modellierungsmächtigkeit
  - begrenzte Unterstützung von Integritätsbedingungen
  - beliebige Namensvergabe • • •

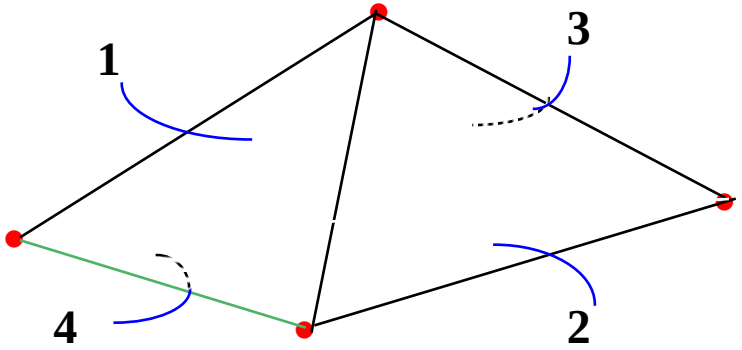
  => Benutzer muß Bedeutung der Daten kennen

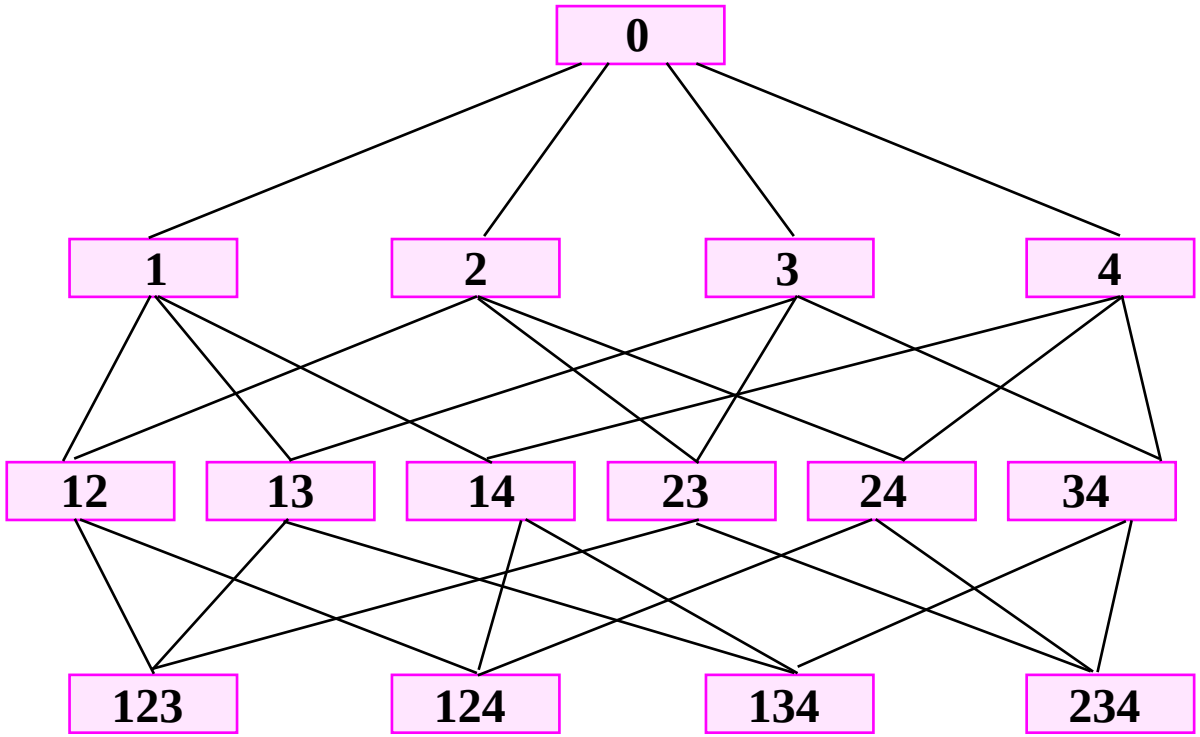
Semantische Datenmodelle (2)

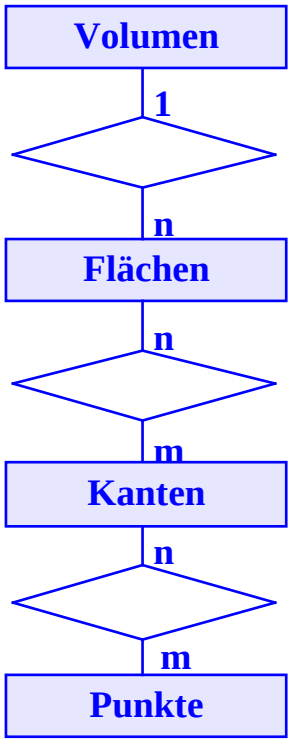
- Grundkonzepte
  - Objekte (Entities) und Objekttypen
  - Beziehungen (Relationships) zwischen Objekttypen
  - Attribute und Wertebereiche
  - Abstraktionskonzepte (meist spezielle Beziehungstypen): Klassifikation, Generalisierung/Spezialisierung, Aggregation
  - Integritätsbedingungen (Schlüsselbedingungen, Disjunktheit zwischen Objekttypen, etc.)
  - Dynamische Konzepte: abgeleitete Daten (Berechnung durch Operation oder Prozedur), Trigger etc.
- Bekannte Vertreter semantischer Datenmodelle:
  - Entity-Relationship-Modell und Erweiterungen
  - statische Modelle mit semantischer Hierarchie (Aggregation, Generalisierung, Gruppierung), Bsp.: SDM, IFO
  - dynamische Modelle mit semant. Hierarchie (Bsp.: TAXIS)
  - irreduzible Datenmodelle (Bsp.: funktionales Datenmodell FDM)
- Haupteinsatzgebiet semantischer Datenmodelle: Unterstützung des logischen DB-Entwurfs (Informationsmodellierung)
  - Betonung der strukturellen Aspekte
  - Operationen werden vernachlässigt

Bsp.: Modellierung von 3D-Objekten im ER-Modell

Beispiel: Tetraeder







Objektorientierte DBS (OODBS)

- Die Entwicklung von OODBS wurde durch neuere DB-Anwendungen vorangetrieben mit weitreichenden Anforderungen bezüglich:
  - Modellierung der Struktur (komplexe Objekte)
  - Modellierung von Verhalten (ADTs, Methoden)

- Auseinanderklaffen von Datenbankmodell und Programmiersprache
  - zwei Welten kommen zusammen
    - "Struktur" wird durch das DBS verwaltet
    - "Verhalten" wird von Anwendungsprogramm (Programmiersprache) nachgebildet
  - "Einbettung" von Datenbank- in Programmiersprache
    - "impedance mismatch" (Fehlanpassung) von DBS und PS
    - unterschiedliche Datentypen
    - Mengen- vs. Satzverarbeitung

- Ziele: Beseitigung des "impedance mismatch"/ bessere Effizienz
  - durch verbesserte Datenmodellierung
  - durch verbesserte Verhaltensmodellierung (Programmentwicklung)

  $\Rightarrow$ Entwurfs- und Wartungseffizienz

  $\Rightarrow$ Ausführungseffizienz

Objektorientierte DBS (2)

- Ansätze zur objektorientierten Datenverwaltung
  - Anreicherung von Programmiersprachen um Persistenz und andere DB-Eigenschaften wie Integrität, Zuverlässigkeit,...) => ***persistente Programmiersprachen***
  - Anreicherung von DBS um objektorientierte Konzepte

  ⇒ ***objekt-relationale DB-Technologie***

- Hochstrukturierte Information (Strukturmodellierung)
  - beliebig zusammengesetzte Einheiten (Typkonstrukteure, Typhierarchien)
  - beliebige Beziehungen zwischen Einheiten
  - Versionen

- Mächtige Operatoren (Verhaltensmodellierung)
  - für typspezifisches Verhalten bei großer Vielfalt der Typen
  - passende Operatoren für spezielle Attributwertebereiche, Versionen, ...

- Unterstützung der Abstraktionskonzepte
  - Klassifikation
  - Generalisierung
  - Aggregation

Objekt-relationale DBS

- Merkmale von ORDBS
  - Erweiterung des relationalen Datenmodells um Objekt-Orientierung
  - benutzerdefinierte Datentypen (u.a. Multimedia-Datentypen)
  - komplexe, nicht-atomare Attributtypen (z.B. relationenwertige Attribute)
  - Bewahrung der Grundlagen relationaler DBS, insbesonderer deklarativer Datenzugriff, Sichtkonzept etc.

- Vergleich
  - relationale DBS: einfache Datentypen, Queries, ...
  - OODBS basierend auf persistenten Programmiersprachen: komplexe Datentypen, gute PS-Integration, hohe Leistung für navigierende Zugriffe
  - ORDBS: komplexe Datentypen, Querying ...

- Standardisierung von ORDBS durch SQL3 bzw. SQL4

Deduktive Datenbanksysteme (DDBS)

- grob wie DBS: verwaltet Fakten (formatierte Daten = extensionale DB)
- zusätzlich: Regeln (intensionale DB) zur Ableitung von implizit vorhandenen Informationen
- Hauptanforderung: effiziente Regelauswertung (Inferenz), Behandlung von Rekursion

Fakten:

F1: Elternteil(C, A) <-
F2: Elternteil (D, A) <-
F3: Elternteil (D, B) <-
F4: Elternteil (G, B) <-
F5: Elternteil (E, C) <-
F6: Elternteil (F, D) <-
F7: Elternteil (H, E) <-

Regeln:

R1: Vorfahr (x, y) <- Elternteil (x, y)
R2: Vorfahr (x, y) <- Elternteil (x, z), Vorfahr (z, y)

Anfrage: ? Vorfahr (x, A)

- auch konventionelle Anwendungen profitieren von rekursiven Anfragemöglichkeiten (Berechnung der transitiven Hülle): Stücklistenauflösung, Wegeprobleme etc.
- Wissensbankverwaltungssysteme (WBVS): DDBS + Verfahren zur Wissensrepräsentation
- Einsatz
  - Unterstützung wissensbasierter Anwendungen
  - Kopplung mit Expertensystemen (XPS)

Information-Retrieval-Systeme (IRS)

Aufgaben/Eigenschaften

- Verwaltung von Dokumenten, Büchern, Abstracts usw.
- effiziente Suche in großen Datenmengen
- typischerweise nur Retrieval im Mehrbenutzerbetrieb

Datenstrukturen

- unformatierte Datenstrukturen: Texte oder Textsurrogate
- Objektbeschreibung: Text in natürlicher Sprache
- Mehrdeutigkeit: Synonym-, Homonymproblem usw.
- Wortstammbildung (Stemming)
- Einsatz eines Thesaurus (komplex organisiertes Wörterbuch):
  Festlegung von Beziehungen zwischen Begriffen, Indexierung von Dokumenten

IRS-Schnittstelle

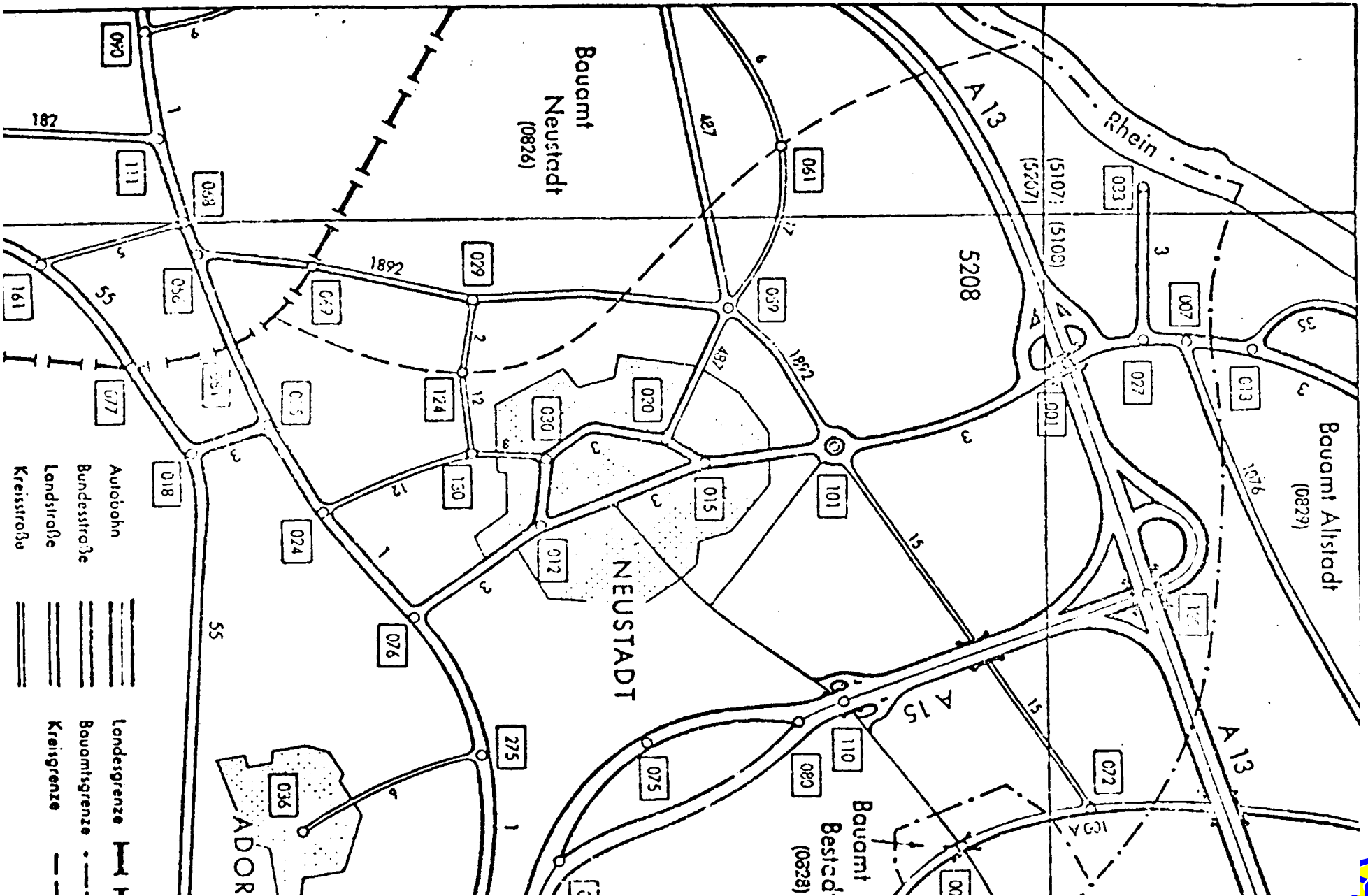
- kein formales Datenmodell
- Dokumentenorganisation oft vorgegeben
- Anfragesprache für Retrieval (=> Annäherung an natürliche Sprache erwünscht)

Information-Retrieval-Systeme (2)

- Art der Suche
  - Anfragen relativ unscharf
  - Ähnlichkeitssuche (nearest neighbor, best match, pattern matching usw.)
  - Ergebnisbewertung: Relevanzproblem (Precision, Recall)

- unterschiedliche Retrieval-Modelle: Boolean, Vektorraum-Ansatz, ...
- Ranking entscheidend für große Ergebnislisten
- zahlreiche Einsatzmöglichkeiten: Bibliotheken, Literaturrecherche, Informationsdienste (Chemie, Recht, Patent-DB, ...
- explosionsartige Zunahme von Anwendungen im World-Wide-Web

Inhalte einer Straßen-DB



Beispiel: Datenbank des Straßeninformationssystems

■ Straßendatenbank ist Bestandsnachweis für das Straßennetz eines Bundeslandes

- Es gibt verschiedene Straßentypen (Autobahnen, Bundesstraßen, Kreisstraßen, Gemeindestraßen etc.)
- Die Straßen sind in Abschnitte eingeteilt, die durch ihre Endknoten definiert sind.
- Straßenabschnitte sind jeweils einem Bauamt zugeordnet
- Die Straßenabschnitte gehen durch Gemeinden. Die Gemeinden gehören zu Kreisen.
- Ein Straßenabschnitt kann mehrere Äste aufweisen (z.B. Aufteilung in 2 Einbahnstrecken). Eine Straße kann an einem Netzknoten unterbrochen sein und an einem anderen Netzknoten weiterführen. Ein Straßenabschnitt kann auf mehreren Straßen (z.B. Bundesstraße und Kreisstraße) gleichzeitig verlaufen.

Topologische Information

- Zusätzlich sind jedem Straßenabschnitt Daten zugeordnet, welche den geometrischen Verlauf zwischen den begrenzenden Netzknoten festlegen (Trassierungselemente: Kreise, Geraden, Klothoiden).
- Die Bauwerke (Brücken, Durchlässe, Signalanlagen etc.) sind dem geometrischen Verlauf des Straßenabschnitts ebenso zugeordnet wie Fußgängerüberwege, Radwege, Gehsteige, Daten des Fahrbahnaufbaus, Höheninformation, Entwässerungsschächte etc.

Unfalldaten, Verkehrsmengen, Frostsicherheit etc. sind weitere Attribute zum Straßenabschnitt

Typische Fragen:

- Auswahl aller Kreisstraßen im Kreis ... mit Breite < 5m und NN-Höhe > 500m.
- Zusammenstellung aller Strecken mit Radwegen getrennt für Ortsdurchfahrt und freie Strecke.
- Auswahl aller Bundesstraßenstrecken im Bauamt ... mit Neigungen größer als 7%.
- Berechnung der befestigten Straßenfläche für alle im Jahr 1980 gebauten Bundesstraßenstrecken

Geo-Informationssysteme (GIS)

- GIS: Computersysteme zur Verarbeitung räumlicher Daten (Erfassung, Speicherung, Manipulation, Analyse, Modellierung, Planung, Darstellung)

- Forderungen
  - DBS-gestützte Datenverwaltung
  - effiziente Verwaltung räumlicher Daten sowie sonstiger Daten
  - breite Einsetzbarkeit
  - graphische Datenerfassung
  - graphische Anfrageformulierung
  - graphische Ergebnisdarstellung

- Räumliche Daten: Verwaltung geometrischer und topologischer Eigenschaften

- Vektordarstellung
  - Grundelemente: Punkt, Linie, Fläche
  - gute Datenstrukturierung
  - geometrische Berechnungen

- Rasterdarstellung
  - flächenmäßige Objektdarstellung durch Pixelmenge
  - einfache Datenerfassung, jedoch geringe Strukturierung
  - große Datenmengen

- Einsatzbereiche: Karthographie, Katasterwesen, Umweltinformationssysteme, Netzinformationssysteme, Raumordnung, kommunale Entwicklungsplanung etc.

Multimedia-Datenbanksysteme

- Verwaltung/Handhabung verschiedener Typen von Information
- Medien: Text, Bilder, Grafik, Ton, Audio, Video, ...
- großer Datenumfang

Medium	Format	Datenumfang	Transferrate
Text	ASCII	1 MB / 500 Seiten	2 KB / Seite
SW-Bilder	G3/4-FAX	32 MB / 500 Bilder	64 KB / Bild
Farbbilder	GIF, TIFF JPEG	1.6 GB / 500 Bilder 0.2 GB / 500 Bilder	3.2 MB / Bild 0.4 MB / Bild
Sprache	MPEG audio	2.4 MB / 5 Min.	8 KB/sec
CD-Musik	CD	52.8 MB/5 Min.	176 KB/sec
Standard-Video	PAL	6.6 GB/5 Min.	22 MB/sec
Qualitätsvideo	HDTV	33 GB/5 Min.	110 MB/sec

- Einsatz von Kompressionstechniken obligatorisch

Multimedia-DBS (2)

- besondere Schwierigkeiten durch zeitabhängige Medien: Sprache, Audio, Video, Animation
  - zeitliche Synchronisation
  - Koordinierung von Ton und Bild bei Video
  - zeitbehaftete Operationen: Playback, Record, Fast-Forward, ...

- Anwendungsklassen
  - Archivsysteme (Zeitung/Radio, Patentamt)
  - Publikation (Büro/Werbung)
  - Überwachung (Prozeßsteuerung, Militär)
  - Unterricht (Animation)

- Verwaltung von Multimediaobjekten innerhalb von DBS?
  - gleichartige Behandlung von einfachen Daten und Multimedia-Objekten
  - Unterstützung von Mehrbenutzerbetrieb
  - Unterstützung von automatischen Recovery-Verfahren und Integritätsbedingungen
  - Vermeidung von Redundanz
  - Geräteunabhängigkeit (CD-ROM, optische Platten etc.)
  - Optimierung der Performance durch System • • •

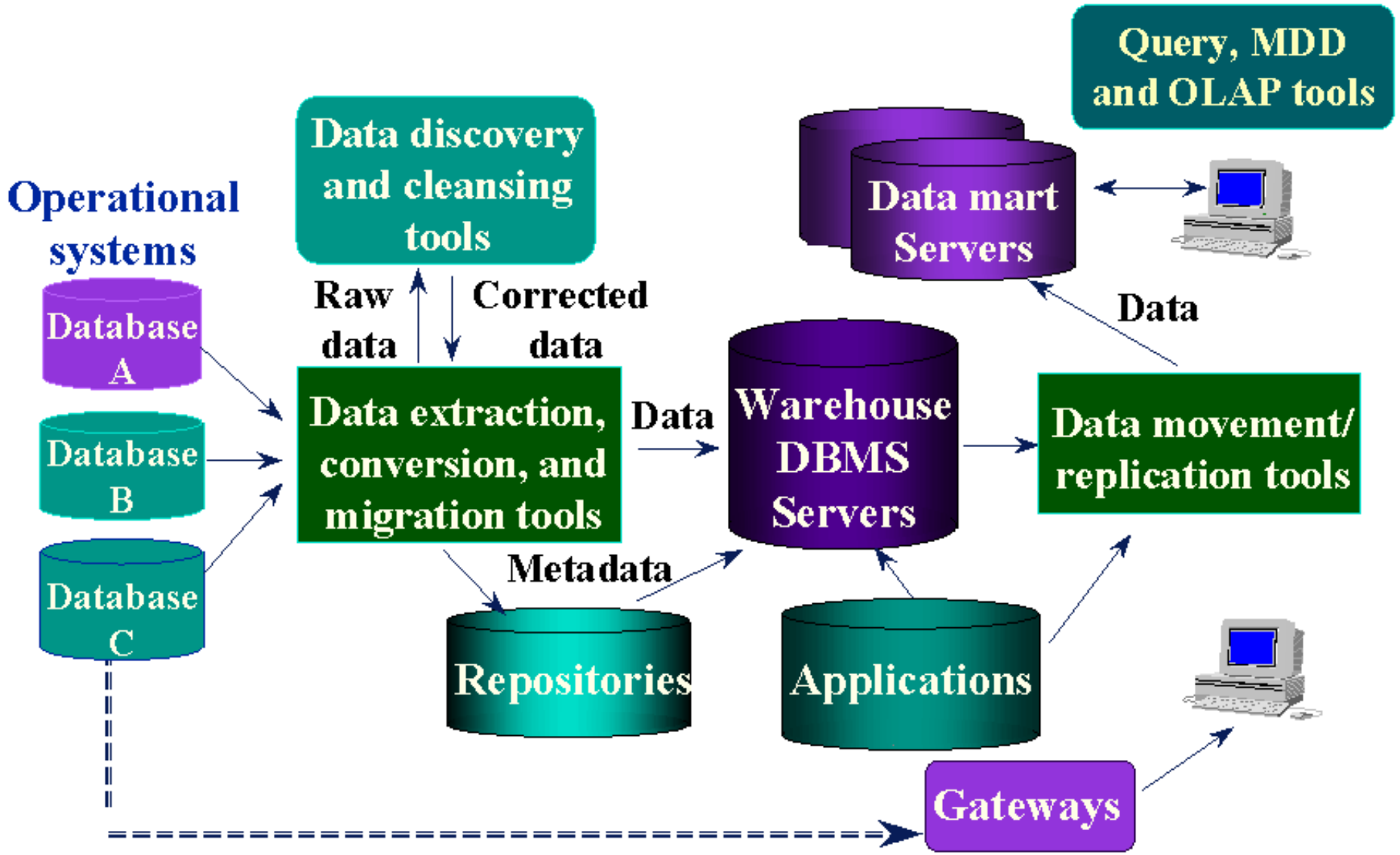
Entscheidungsunterstützende Systeme (Decision Support Systems, DSS)

- **OLAP** (Online Analytical Processing) vs. OLTP (Online Transaction Processing)
  - Analyse betrieblicher Datenbestände

- **Data Mining**: Aufspüren von inhärenten Daten-/Informationsmustern aus großen dynamischen Datenbeständen
  - "In data mining applications, not only does the system define the semantics, it actually defines the queries. The user simply says "Go", and the system produces what it believes to be useful answers"
  - oft synonym: KDD (Knowledge and Data Discovery)

- häufiger Einsatz von **Data Warehouses**
  - Bereitstellung der Datenbestände eines Unternehmens für den Endbenutzer, so daß nicht nur ein vorgegebener Blickwinkel auf die Daten unterstützt wird
  - Datenbestand und aufbauende Werkzeuge für nahezu alle anfallenden Fragestellungen
  - Bsp.: Umsatzentwiclung nach Zeit, Produktklasse, Region, etc.

Data Warehouse-Umfeld



Zusammenfassung

- DBS-Technologie
  - Verwaltung von persistenten Daten
  - effizienter Datenzugriff
  - flexibler Mehrbenutzerbetrieb
  - Konzepte: **Datenmodell und DB-Sprache**, Transaktion als Kontrollstruktur
- Relationenmodell zur Unterstützung von anspruchsvollerer Anwendungen (GIS, CAD, Multimedia) unzureichend
- Einteilung unterschiedlicher DBS-Klassen:

Anwendungsbereich	**Standard-Anwendungen**	**OLAP**	**GIS**	**CAD/ CASE**	**Multimedia-Anw.**	**XPS**
Datenmodell	**RM**	**ORDBS**			**OODBS**	**WBS**
Architektur	**zentralisierte DBS**	**Verteilte/ Föderative DBS**			**Parallele DBS**	**C/S-DBS**

2. Gundkonzepte von Objektorientierten DBS

- Einführung
  - Objektorientierung
  - Klassifikation von Datenmodellen
  - Definition von OODBS
- Objektidentität
- Komplexe Objekte
- Kapselung/Abstrakte Datentypen
- Klassenhierarchie/Vererbung
- Überladen/spätes Binden
- Operationale Vollständigkeit
- Weitere Eigenschaften
  - BLOBs
  - Versionen

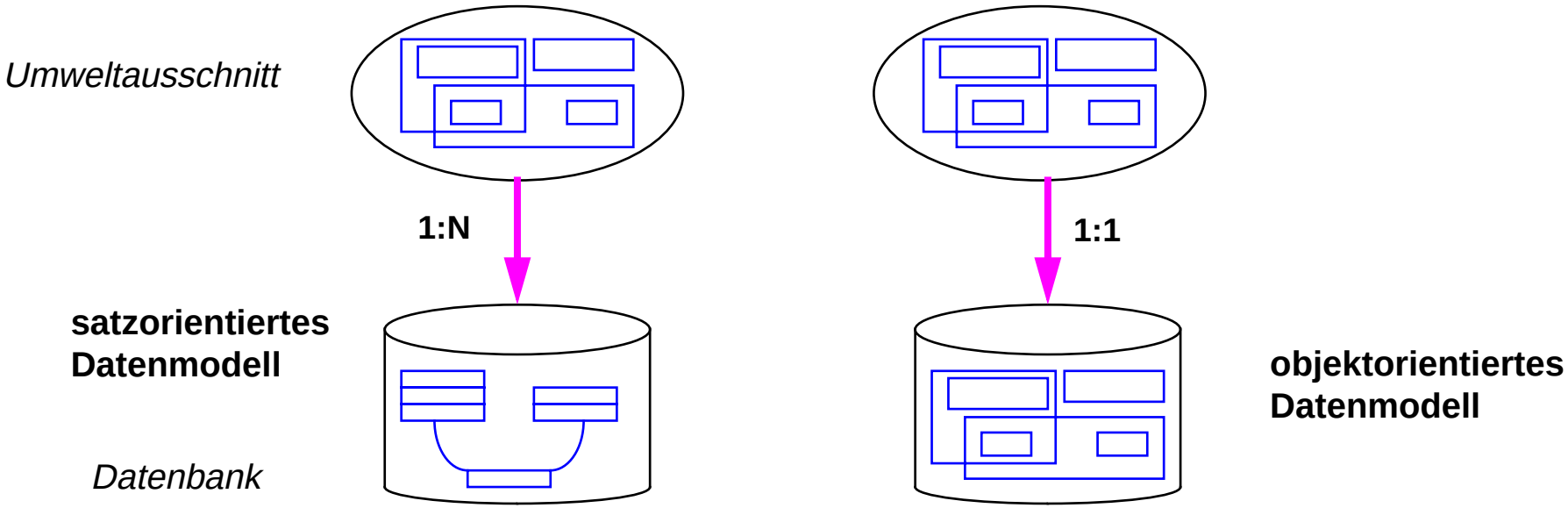
Anforderungen an 'Next-Generation DBMS'

- Unterstützung nicht-konventioneller Einsatzbereiche:
  - CAD/CAM
  - CASE
  - Bildverarbeitung
  - Robotik, Realzeitanwendungen
  - Büroinformationssysteme, ...

- Neue Anforderungen:
  - Komplexe Objekte
  - Abstrakte Datentypen
  - Erweiterbarkeit
  - Abstraktionskonzepte
  - Rekursion
  - komplexe Integritätsbedingungen
  - Versionen
  - neue (erweiterte) Transaktionskonzepte
  - mehrdimensionale Speicherungsstrukturen

  •••

Objektorientierung

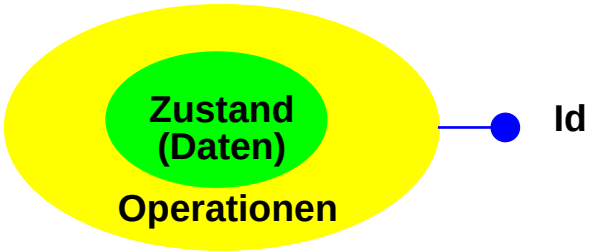


- Suche von Objekten ?
- Wirkung von Aktualisierungs-Operationen (Einfügen, Löschen, Kopieren ...) ?
- Erhaltung der Konsistenz ?
- Performance ?

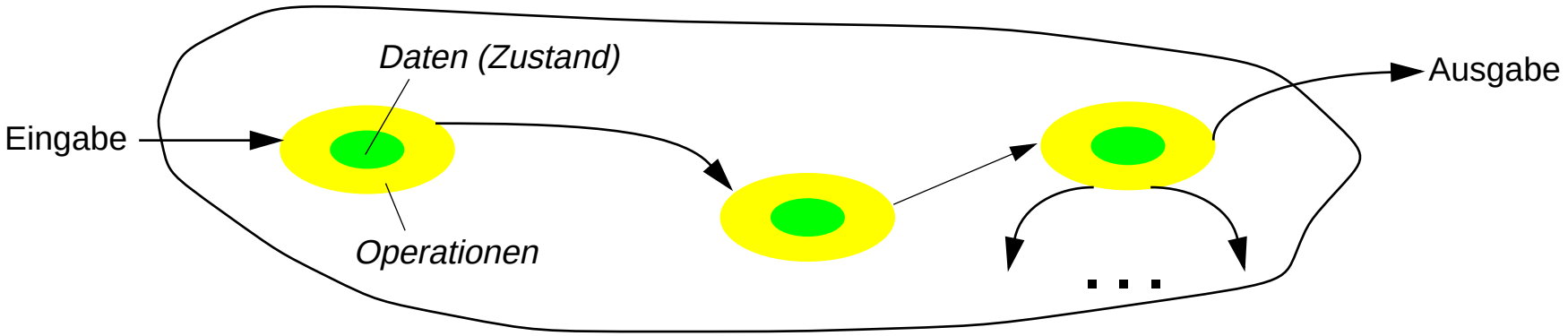
Objektorientierung (2)

- "Alles" ist ein Objekt?

- Objekte haben:
  - Identifikator
  - internen Zustand, der mit Hilfe von Attributen beschrieben wird
  - Menge von Operationen (Methoden), die ihre Schnittstelle zur externen Welt definieren



- Objekte kommunizieren miteinander über Methodenaufrufe (Nachrichten)



Klassifikation von Datenmodellen

satzorientiertes Datenmodell

- einfache Objekte aus atomaren oder zusammengesetzten Elementtypen
  - ➥ begrenzte, feste Anzahl von Zusammensetzungsstufen
  - ➥ keine Definition beliebiger neuer Typen

+

- ausschließlich vordefinierte, generische Operatoren
  - ➥ Tupel erzeugen und löschen, Attributwerte modifizieren, Suche nach Werten
  - ➥ Operationen der DB-Sprache

strukturell objektorientiertes Datenmodell (Komplex-Objekt-Modell)

- zusammengesetzte Objekte (strukturiert, komplex)
  - ➥ keine Beschränkung der Zusammensetzungsstufen
  - ➥ ggf. Modellbeschränkungen: Überlappung, Rekursion, expl. Beziehungen

+

- vordefinierte (generische, ein-/mehrstufige) Operatoren

Klassifikation von Datenmodellen (2)

verhaltensmäßig objektorientiertes Datenmodell

- satzorientiert hinsichtlich Elementaufbau

+

- Möglichkeiten zur Definition typspezifischer Operatoren
  - ➥ benutzer-orientierte Operatoren auf satzorientierten Strukturen
  - ➥ Codierung benutzt Wertestruktur des Typs (Repräsentation) und vordefinierte Operatoren (gemäß Struktur) und/oder bereits verfügbare andere benutzerdefinierte Operatoren

voll objektorientiertes Datenmodell

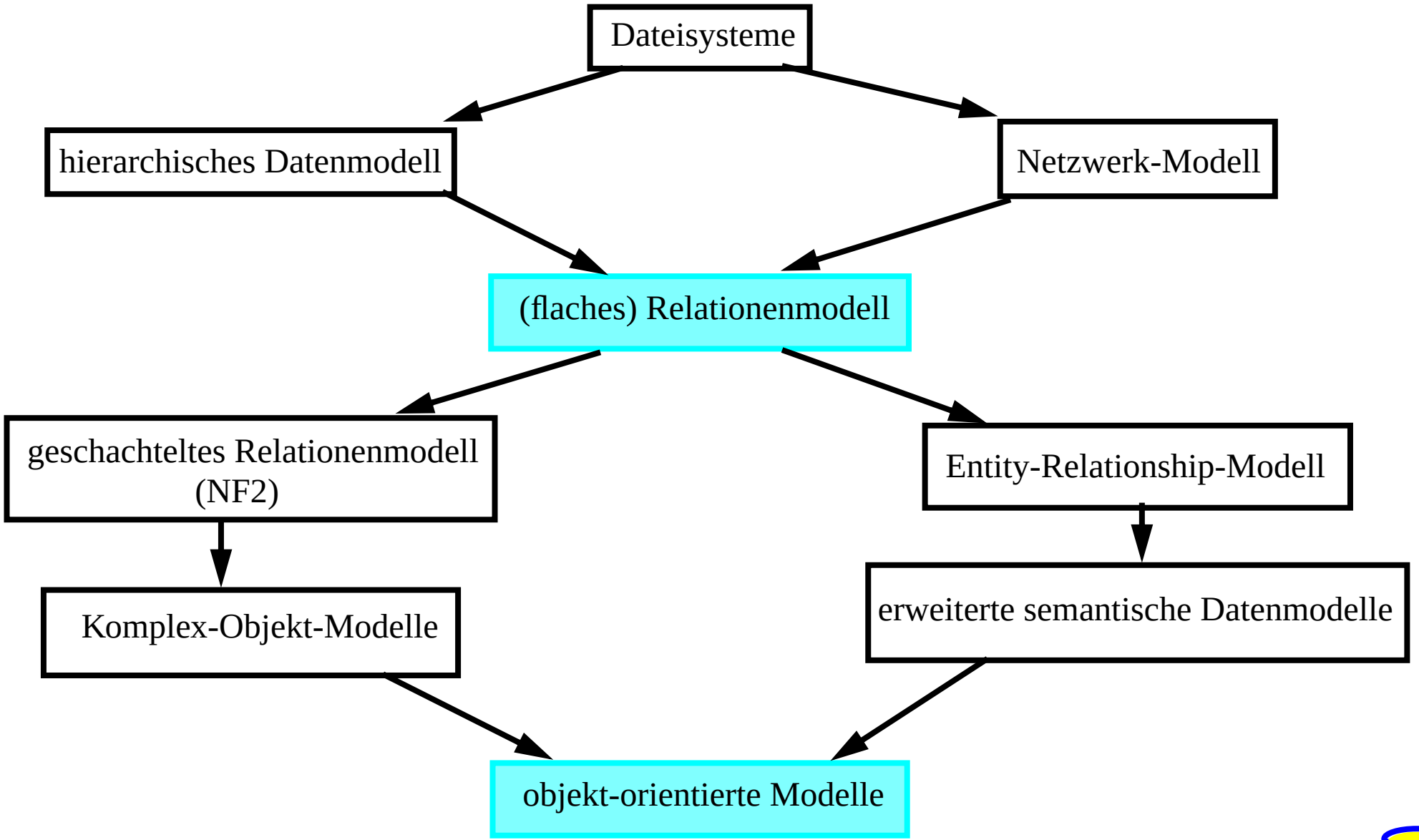
- strukturell objektorientiert

+

- verhaltensmäßig objektorientiert
-

⇒ komplexe Strukturen + definierbares Verhalten

Entwicklung von Datenmodellen



Definition eines objektorientierten DBS*

- ein objekt-orientiertes DBS (OODBS) muß zwei Kriterien erfüllen
  - es muß ein DBS sein
  - es muß ein objektorientiertes System sein

- DBS-Aspekte:
  - Persistenz
  - Externspeicherverwaltung
  - Datenunabhängigkeit
  - Transaktionsverwaltung (Synchronisation, Logging, Recovery)
  - Ad-Hoc-Anfragesprache

- essentielle OOS-Aspekte:
  - Objektidentität
  - komplexe Objekte
  - Kapselung
  - Typ-/Klassenhierarchie
  - Vererbung
  - Überladen und spätes Binden
  - operationale Vollständigkeit
  - Erweiterbarkeit

- optionale OOS-Aspekte:
  - Mehrfachvererbung
  - Versionen
  - Design-Transaktionen •••

* M. P. Atkinson, et. al, “The Object-Oriented Database System Manifesto”, Proc. Deductive and Objekt-Oriented Databases, Elsevier 1989

OOS-Aspekte

1. Objektidentität

- Existenz des Objektes ist unabhängig von seinem Wert
- Objekte können identisch (dasselbe Objekt) oder gleich (derselbe Wert) sein

2. Komplexe Objekte

- Anwendung von Objekt-Konstruktoren auf einfache Objekte
- minimal drei Konstruktoren: Mengen, Listen, Tupel

3. Kapselung

- Trennung von Schnittstelle und Implementierung (→ ADTs)
- Schnittstelle: Spezifikation einer Menge von Operationen (Methoden); sichtbar
- Implementierung: Datenteil (Repräsentation des Objektzustandes) und Prozedurteil (Operationen)

4. Typen oder Klassen (oft Synonyme)

- Typ beschreibt gemeinsame Struktur einer Menge von Objekten mit gemeinsamen Charakteristika --> ADT
- Klasse: Spezifikation ist die gleiche wie die des Typs; zusätzlich umfaßt sie Laufzeitaspekte: Gruppierung der Instanzen (Kollektion)

OOS-Aspekte (2)

5. Vererbung

- Klassen und Typen können in Vererbungshierarchien organisiert werden
- Spezialisierung in Subklassen und Subtypen
- Spezialisierte Klassen/Typen erben Struktur, alle Methoden, Integritätsbedingungen und ggf. Defaultwerte von allgemeineren Klassen/Typen
- Subklassen können zusätzlich eigene Attribute, Methoden und Integritätsbedingungen besitzen

6. Überladen und spätes Binden

- Methoden können für Subklassen redefiniert werden (overriding)
- Name der Methode bleibt gleich, Wirkung der Methode ist objektabhängig
- Überladen impliziert Binden zur Laufzeit (late binding)

7. Operationale Vollständigkeit

- jede berechenbare Funktion kann unter Benutzung des Systems programmiert werden
  - ➥ größere Funktionalität als herkömmliche Teilsprachen

8. Erweiterbarkeit

- vordefinierte Typen oder Klassen können erweitert werden
  - ➥ neue Typen können definiert werden
- kein Unterschied zwischen system- und benutzerdefinierten Typen

Objektidentität

- Relationenmodell ist "wertebasiert"
  - Identifikation von Daten durch Schlüsselwerte
  - Benutzer muß häufig künstliche Schlüsselattribute einführen
  - Repräsentation von Beziehungen führt zu erheblicher Redundanz (Fremdschlüssel)
  - schwierige (ineffiziente) Modellierung komplexer Strukturen
  - Änderungsprobleme (referentielle Integrität, Sichtkonzept)

- OODBS: Objekt = (OID, Zustand, Operationen)

- Objektidentität
  - systemweit eindeutige Objekt-Identifikatoren
  - OID während Objektlebensdauer konstant, üblicherweise systemverwaltet
  - OID tragen keine Semantik (<-> Primärschlüssel im RM)
  - Änderungen beliebiger Art (auch des Primärschlüssels im RM) ergeben *dasselbe* Objekt

- Notwendigkeit künstlicher Primärschlüssel wird vermieden

Objektidentität (2)

- Realisierung von Beziehungen über OIDs

```
ABT  (OID:  IDENTIFIER, (* implizit *)
      ANAME:  STRING,
      LEITER: PERS ... (* Referenz-Attribut *)
```

```
PERS   (OID: IDENTIFIER (* implizit *),
        PNAME: STRING,
        ABTLG: ABT, (* Referenz-Attribut *)
```

- gemeinsame Teilobjekte ohne Redundanz möglich (referential sharing)
- Wartung der referentiellen Integrität wird erleichtert
- implizite Dereferenzierung über *Pfadausdrücke* anstatt expliziter Verbundanweisungen

```
SELECT   PNAME, ABTLG.LEITER.PNAME
FROM     PERS
WHERE ABTLG.ANAME = "Forschung"
```

Objektidentität (3)

- Identität vs. Gleicheit
  - zwei Objekte sind identisch (O1 == O2), wenn sie dieselbe OID haben
  - zwei Objekte sind gleich (O1 = O2), wenn sie den gleichen Zustand besitzen
  - beides sollte ausdrückbar sein

- ermöglicht eindeutige Semantik von Änderungsoperationen

Name	Gehalt
Müller	50K
Meier	70K

Name	Gehalt
Müller	60K

Objekte vs. Werte

Objekte	Werte
nicht druckbar	druckbar
anwendungsabhängige Abstraktion	anwendungsunabhängige Abstraktion
müssen erzeugt und definiert werden	müssen nicht erzeugt oder definiert werden
tragen selbst keine Information	tragen selbst die Information
werden beschrieben	beschreiben etwas

- Beispiel für Objekte: Personen, Bücher, etc.
- Werte: Eigenschaften (z.B. Geburtsdatum), 5, 3, ...

Komplexe Objekte

- Relationenmodell
  - nur einfache Attribute, keine zusammengesetzte oder mengenwertige Attribute
  - nur zwei Typkonstruktoren: Bildung von Tupeln und Relationen (Mengen)
  - keine rekursive Anwendbarkeit von Tupel- und Mengenkonstruktoren

- OODBS: Attribute können sein:
  - Standardtypen (Integer, Char, ...)
  - zusammengesetzte Typen
  - benutzerdefinierte Typen (Zeit, Datum etc.)
  - andere Objekte (über Referenzen auf solche)

- Erzeugen strukturierter (zusammengesetzter) Datentypen aus Basistypen durch zusätzliche Typkonstruktoren
  - RECORD/TUPEL
  - SET (Mengenbildung)
  - LIST/SEQUENCE (Reihenfolge!)
  - ARRAY-Konstruktor (VECTOR)
  - MULTISET / BAG (Duplikate zugelassen)

    ➥ beliebige (rekursive) Kombination aller Konstruktoren zum Aufbau komplexer Objekte

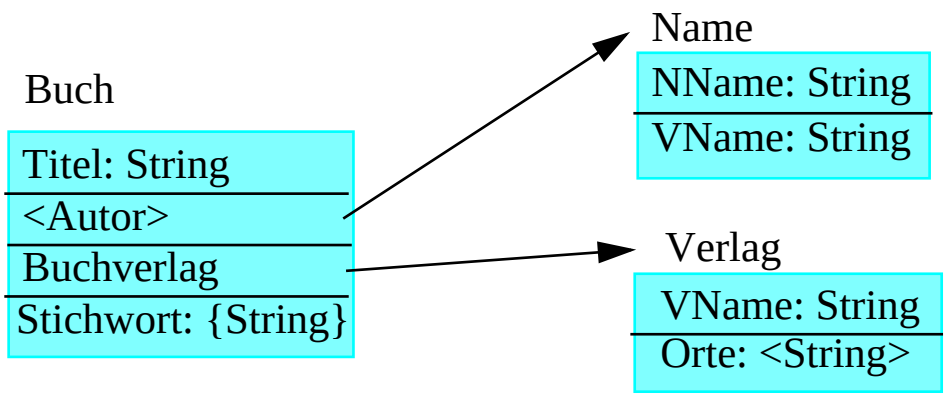
Komplexe Objekte (2)

- Ein OODBS sollte wenigstens unterstützen
  - den Typkonstruktor **TUPLE [ ]** sowie die *Kollektionstypen* **LIST** < > und **SET { }** sowie
  - ihre beliebige Kombination

- Beispiel: Objektmenge "Buch“

```
BUCH {      [     Titel:        String,
                  Autor:         < Name: [ NName:  String, VName: String ] >
                  Buchverlag:   VERLAG, (* Referenzattribut *)
                   Stichwort:    { Swort: String }
            ]   }
```

- graphische Darstellung



Komplexe Objekte (3)

- Ausprägungen (Beispiel)

Buch

	Titel	Autor	Buch-verlag	Stichwort
#10	Mehrrechner-DBS	<#21>	#31	{DBS, Verteilung, Parallelität}
#11	Datenbankeinsatz	<#22, #23>	#32	{Modellierung, DBS}

Name

	NName	Vname
#21	Rahm	Erhard
#22	Lang	Stefan
#23	Lockemann	Peter

Verlag

	VName	Orte
#31	Addison-Wesley	<Bonn, Paris, Reading, New York>
#32	Springer	<Berlin, Heidelberg, New York>

- Generische Operationen auf Tupel-/Kollektionstypen: Komponentenzugriff, MEMBER, FIRST, ...

- Anfragebeispiele
  - Titel aller Bücher zum Stichwort "Datenbanksysteme"
  - Titel aller Bücher mit "Rahm" als erstem Autor

Kopieren komplexer Objekte

- Flaches Kopieren (shallow copy): neues Objekt erhält gleichen Zustand wie zu kopierendes Objekt

Buch-Beispiel:

	Titel	Autor	Buchverlag	Stichwort
#10	Mehrrechner-DBS	<#21>	#31	{DBS, Verteilung, Parallelität}

shallow copy

	Titel	Autor	Buchverlag	Stichwort
#15	Mehrrechner-DBS	<#21>	#31	{DBS, Verteilung, Parallelität}

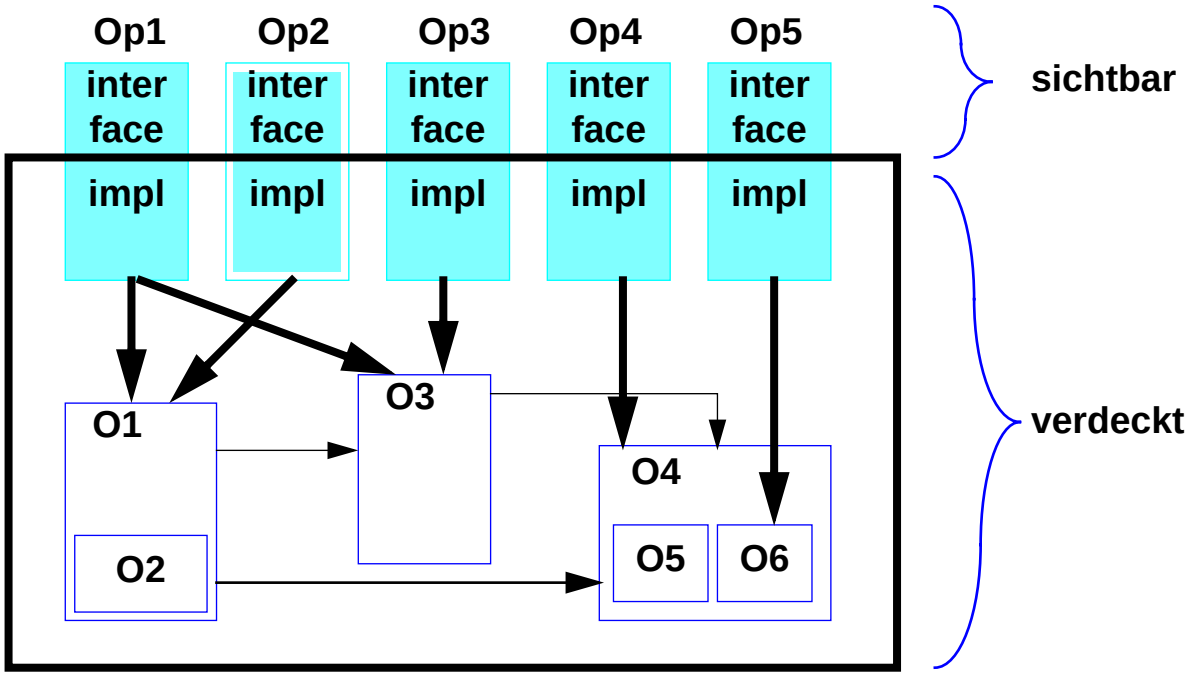
- Tiefes Kopieren (deep copy): Komponentenobjekte werden nicht übernommen, sondern kopiert (rekursiv über alle Stufen).

	Titel	Autor	Buchverlag	Stichwort
#16	Mehrrechner-DBS	<#25>	#34	{DBS, Verteilung, Parallelität}

- Unterschiedliche Gleichheitstests: Shallow Equal vs. Deep Equal

Kapselung von Objekten

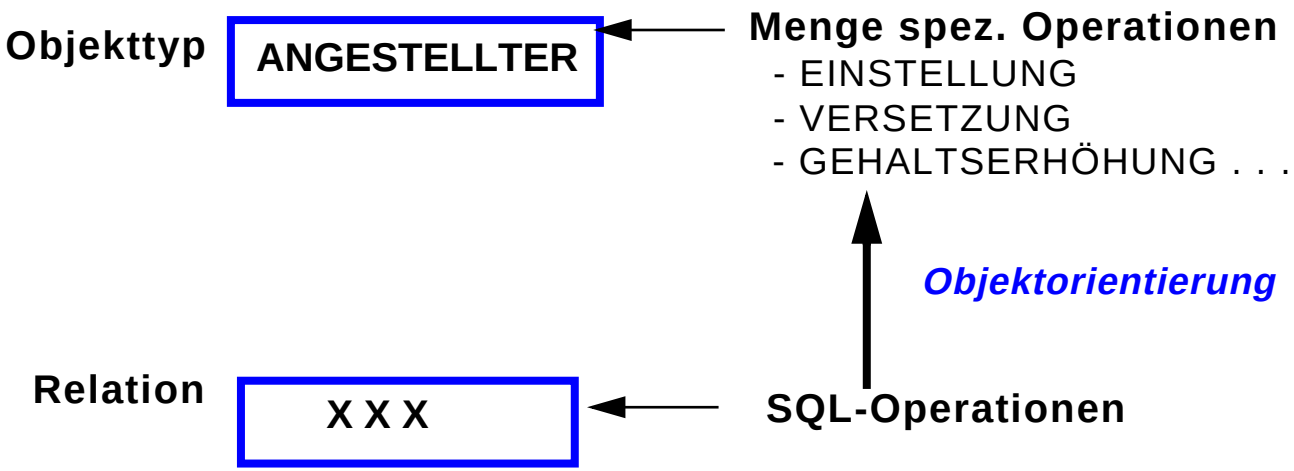
- Repräsentation von Objektinstanzen bleibt unsichtbar
- Anwender sieht nur Operatorschnittstellen (“Signaturen”)



- Objektzugriff ist ausschließlich mit den definierten Operatoren möglich
- Code der Operatoren bleibt Anwender verborgen

Abstrakte Datentypen (ADT)

- Objekt = Struktur + Verhalten
- Geheimnisprinzip (Information Hiding)
  - Struktur des Objektes ist verborgen
  - Verhalten des Objektes ist ausschließlich durch seine *Operationen* (Methoden) bestimmt
  - nur Namen und Signatur (Argumenttypen, Ergebnistyp) von Operationen werden bekannt gemacht
  - Implementierung der Operationen bleibt verborgen



- Kapselung erhöht
  - (logische) Datenunabhängigkeit
  - Datenschutz

Abstrakte Datentypen (2)

- rekursive Anwendbarkeit des ADT-Konzeptes auf Teilobjekte und Attribute
- Erzeugung problembezogener Datentypen als neue Basistypen
- zugeschnittene Operatoren und Funktionen
- Beispiel:

  ADT *DATE* , Operator '-'

```
SELECT   'Beschäftigungsdauer:'  HEUTE ()  -  P.EDATUM
FROM     PERS P
```

- Weitere Anwendungsbeispiele: COMPLEX, MATRIX, TEXT, IMAGE, LINIE • • •
- strikte Kapselung jedoch oft nicht gewünscht
  - eingeschränkte Flexibilität
  - mangelnde Eignung für Ad-Hoc-Anfragen
  - Koexistenz von attributbezogenen Anfragen und objekttypspezifischen Operationen

Operationen

- Generelle Klassen:
  - Konstruktoren (Erzeugung/Initialisierung neuer Instanzen eines Objekttyps)
  - Destruktoren
  - lesende Operationen (Beobachter)
  - Mutatoren
- Vordefinierte vs. benutzerdefinierte Operationen
- Beispiele für vordefinierte Operationen:
  - Erzeugen/Löschen von Objektinstanzen
  - generische Funktionen auf Kollektionen (Mengen, Listen, Arrays)
  - Komponentenzugriff
- Verwaltung der Operationen im DBS
  - Reduzierung des "impedance mismatch"
  - verringerte Kommunikationshäufigkeit zwischen Anwendung und DBS

Anwendungsbeispiel: Verwaltung räumlicher Objekte

- Relationenmodell bietet keine Unterstützung
- hohe Komplexität bereits in einfachen Fällen
- Beispiel: Darstellung von Rechtecken in der Ebene

a) Relationenmodell

R-ECK (RNR, X1, Y1, X2, Y2: INTEGER)

b) ADT-Lösung

ADT **BOX** mit Funktionen
INTERSECT, CONTAINS, AREA, usw.

R-ECK (RNR: INTEGER, Beschr: BOX)

Finde alle Rechtecke, die das Rechteck ((0,0) (1,1)) schneiden!

```
SELECT RNR FROM R-ECK
WHERE (X1 > 0 AND X1 < 1 AND Y1 > 0 AND Y1 < 1)
      OR (X1 > 0 AND X1 < 1 AND Y2 > 0 AND Y2 < 1)
      OR (X2 > 0 AND X2 < 1 AND Y1 > 0 AND Y1 < 1)
      OR (X2 > 0 AND X2 < 1 AND Y2 > 0 AND Y2 < 1)
      OR . . .
```

```
SELECT RNR FROM R-ECK
WHERE INTERSECT (Beschr, (0, 0, 1, 1))
```

Typen/Klassen

- Typen und Klassen werden teilweise synonym, teilweise in unterschiedlicher Bedeutung verwendet

- Typ
  - Festlegung von Struktur (Wertebereich) und Operationen für Objekte
  - Typen sind meist während der Laufzeit nicht mehr veränderbar
    (=> ermöglicht Typprüfung zur Übersetzungszeit)
  - intensionale Darstellung
  - Abstrakter Datentyp: besitzt benutzerdefinierte Methoden, unterstützt Kapselung

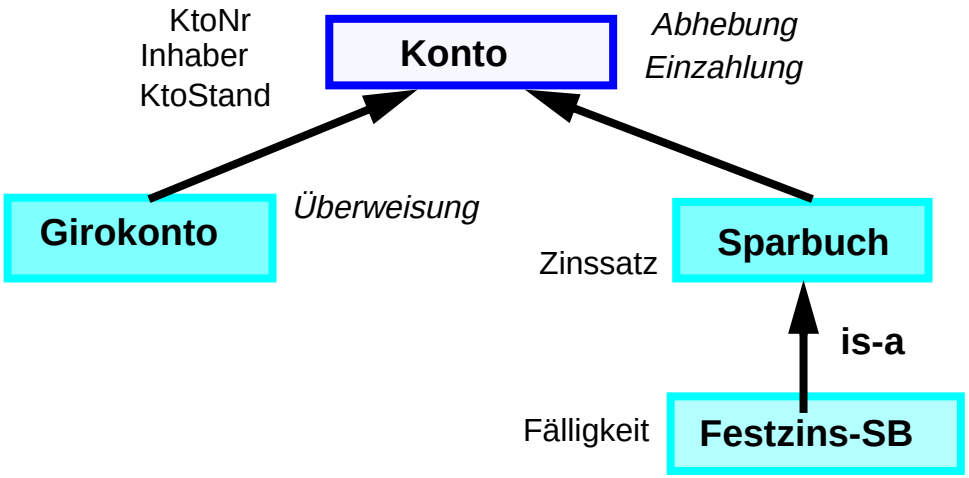
- Klasse
  - Behälter von Objekten (Instanzen) mit spezifischen Eigenschaften
  - Klassen sind während Laufzeit änderbar
  - extensionale Darstellung
  - manchmal: im Vergleich zu Typen steht hier die Erzeugung neuer Instanzen der Klasse im Vordergrund

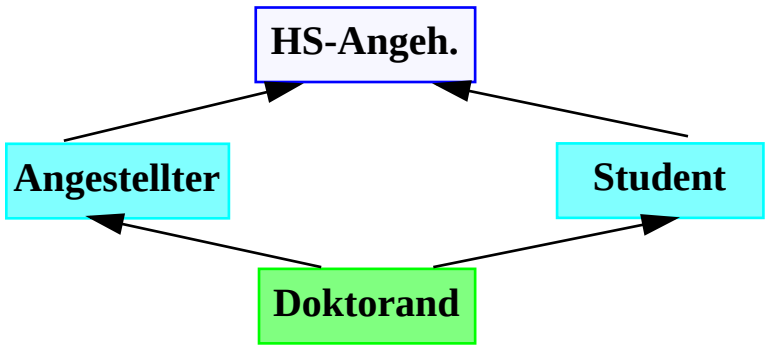
- andere Sichtweise (Lausen/Vossen96): Klasse *hat* Typ zur Festlegung der Wertebereiche für den Zustand; Festlegung des Verhaltens (benutzerdefinierte Methoden) nicht Teil des Typs

- Typen und Klassen können in einer Generalisierungshierachie organisiert werden

Typ/Klassen-Hierarchie

- Anordnung von Objekttypen in Generalisierungs-/Spezialisierungshierarchie (IS-A-Beziehung)
- Vererbung von Attributen, Methoden, Integritätsbedingungen und Default-Werten
- Arten der Vererbung: einfach (Hierarchie) vs. mehrfach (Typverband)





Typ/Klassen-Hierarchie (2)

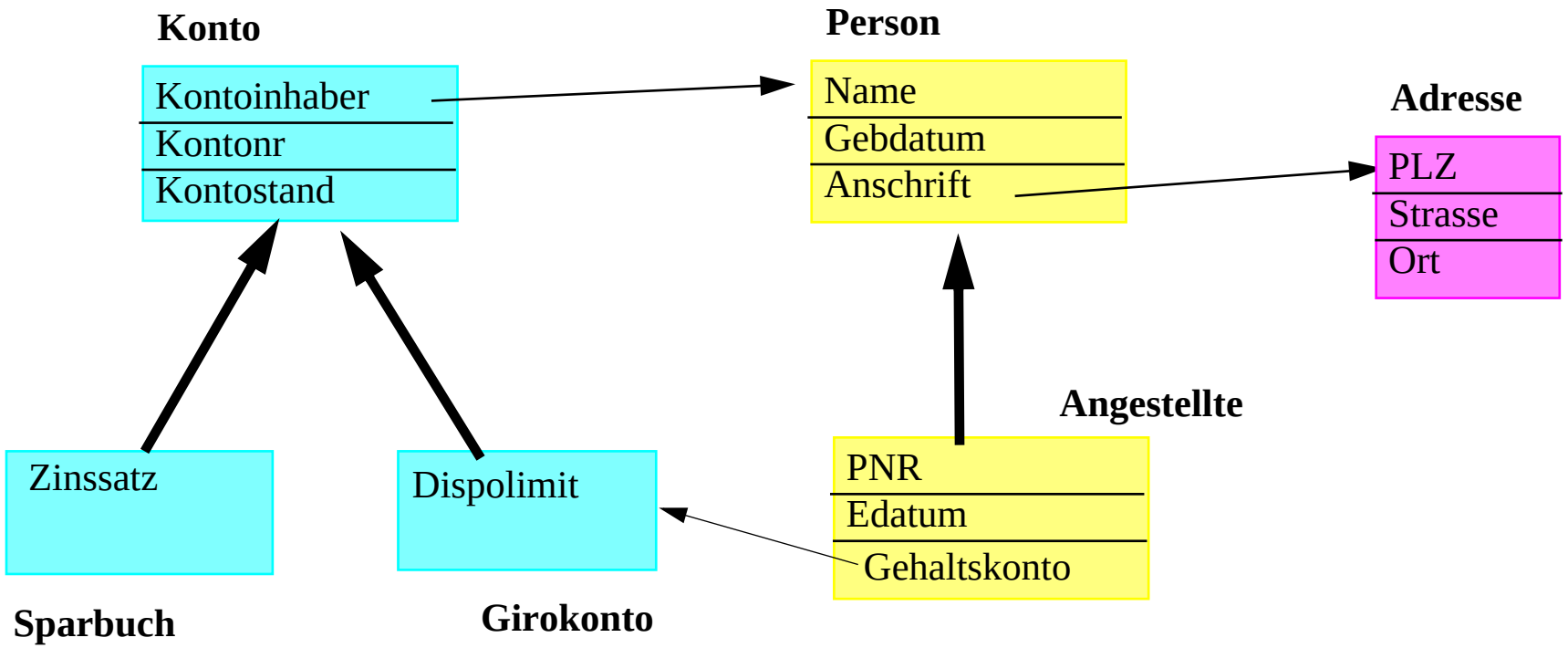
- Prinzip der Substituierbarkeit: Instanz einer Subklasse B kann in jedem Kontext verwendet werden, in dem Instanzen der Superklasse A möglich sind (jedoch nicht umgekehrt)
  - Methoden der Oberklasse sind auch für Objekte einer Unterklasse ausführbar
  - Zuweisungsregel: einer Oberklasse können Objekte von Unterklassen zugewiesen werden

```
define f (x: A)
a: A;  b: B;
a := b;
f (b);
```

- Substituierbarkeitsprinzip impliziert, daß Klasse heterogene Objekte enthalten kann

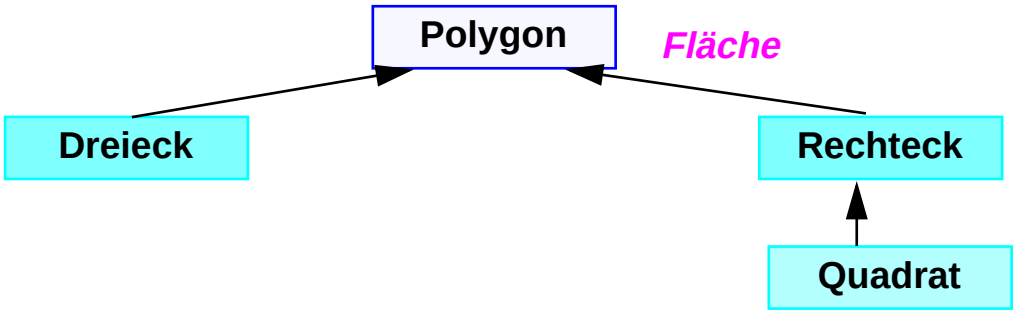
- Vorteile des Vererbungsprinzips:
  - Code-Wiederverwendung (reusability)
  - Erweiterbarkeit
  - Repräsentation zusätzlicher Semantik
  - Modellierungsdisziplin (schrittweise Verfeinerung)

Kombination von Generalisierungs- und Aggregationshierarchien



Überladen (Overloading)

- Overloading: derselbe Methodenname wird für unterschiedliche Prozeduren verwendet
- Overloading kann auch innerhalb von Klassenhierarchien angewendet werden



  - Redefinition von Methoden für Subklassen (Overriding)
  - spezialisierte Methode mit gleichem Namen

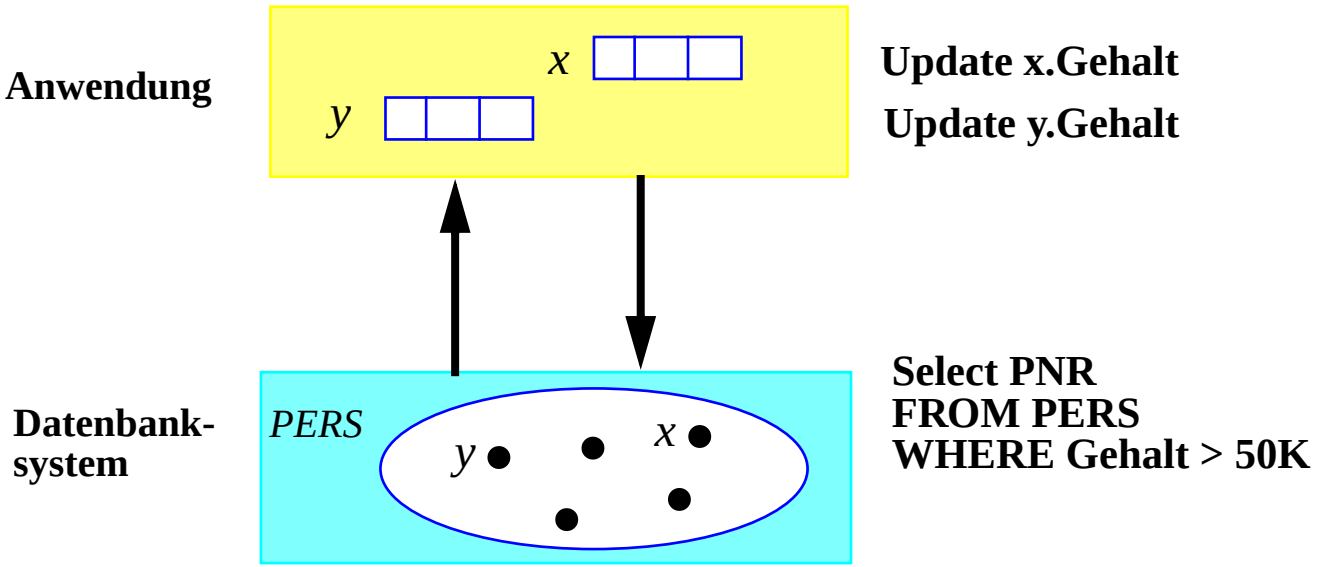
- Verwendung eines einzigen Funktionsnamens erleichtert Realisierung nutzender Programme und verbessert Software-Wiederverwendbarkeit
- Überladen impliziert dynamisches (spätes) Binden zur Laufzeit (late binding)
- Polymorphismus (griech. "viele Formen"): gemeinsame Schnittstelle für unterschiedliche Operationen
  - polymorphe Methoden: Name der Methode bleibt gleich, Wirkung der Methode ist objektabhängig
  - Variante: generischer Polymorphismus (Objekttyp wird als Parameter übergeben)

Operationale Vollständigkeit (computational completeness)

- nicht alle Berechnungen in herkömmlichen DB-Anfragesprachen möglich (DML ist Teilsprache)

  => Einbettung in allgemeine Programmiersprache

- Verwendung zweier Sprachen führt zu *Impedance Mismatch*:
  - Fehlanpassung von Datenbanksprachen (DDL/DML) und herkömmlichen Programmiersprachen
  - unterschiedliche Typ-Systeme (nur begrenzte Typ-Prüfungen möglich, Typkonversionen)
  - deklarative DML vs. prozedurale PS
  - mengen- vs. satzorientierte Verarbeitung => Cursorkonzept
  - umständliche, fehleranfällige Programmierung



Operationale Vollständigkeit (2)

- Alternative: einheitliche DB-Programmiersprache (DBPL)
  - persistente Datenstrukturen
  - vollständiges und minimales Typsystem, erweiterbar
  - objektorientierer Zugriff
  - auf Virtuellen Speicher ausgerichtet
- Abstraktion eines einstufiges Speichermodells
  - transparente Abbildung auf physische Speicher (RAM, Externspeicher)
  - Unterstützung von Anfragen, Transaktionsverwaltung und allgemeinen Berechnungen auf beliebigen Datenstrukturen
- Vorteile
  - nur eine Programmierschnittstelle
  - keine Konversionen von Datentypen
  - operationale Vollständigkeit
- Probleme
  - fehlende Mehrsprachenfähigkeit
  - geringer Schutz (offen, gleicher Adreßraum)
  - mengenorientierte Verarbeitung

Verwaltung von Multimedia-Objekten

- integrierte Verwaltung von Text-, Bild-, Audio-, Video- ... Daten

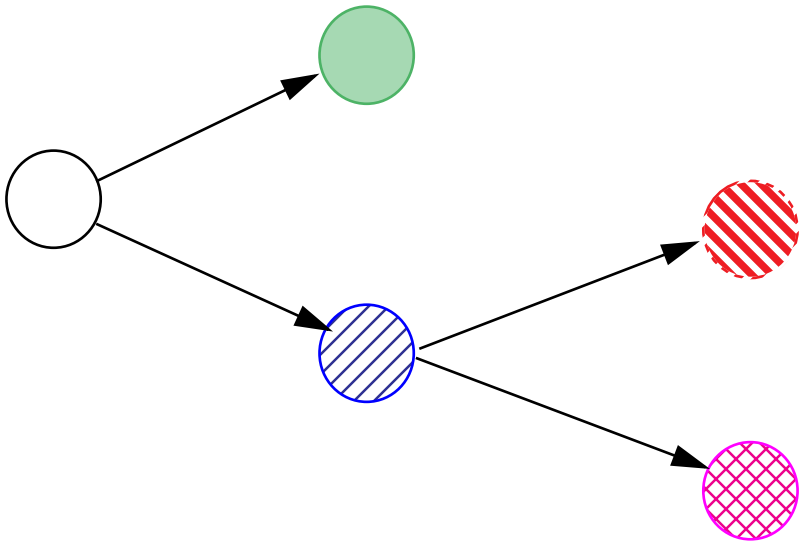
- Einsatz von BLOBs (Binary Large Objects) bzw. "long fields"
  - Speichern von großen, unstrukturierten Bytefolgen (z.B. Rasterbilder)
  - i.a. nur einfache Operationen erforderlich

- Einsatz von Multimedia-ADT
  - zugeschnittene Operationen
  - Verwendung von ADT-Funktionen in DB-Anfragen
  - explizite Repräsentation von Objektstruktur (z.B. bei Texten/Dokumenten und Geo-Objekten)

- hohe Leistungsanforderungen an DBMS
  - Erweiterbarkeit des DBS um neue Datentypen und Operationen
  - große Datenmengen, komplexe Operationen
  - Erweiterbarkeit erforderlich bei Query-Optimierung, Indexstrukturen, Pufferverwaltung, Externspeicherverwaltung ...
  - parallele Anfrageverarbeitung auf neuen Datentypen

Unterstützung von Versionen und Alternativen

- wichtig v.a. für Entwurfsumgebungen (CAD, CASE)
- Verwaltung von Versionsgraphen



- *Konfiguration*: Menge der Versionen, die ein aktuelles Produkt bilden (-> Konfigurationsverwaltung)
- zahlreiche Vorschläge

Zusammenfassung

- OODBS unterstützen
  - komplexe Objekte und mächtige Operationen
  - Klassenhierarchien und Vererbung
  - Erweiterbarkeit
- OODBS eignen sich für Non-Standard-Anwendungen
- Benutzer hat zwischen Objekten und Datenwerten zu unterscheiden
- Es gibt nicht ein objektorientiertes Datenmodell, sondern viele!
  - erhebliche Abweichungen zwischen OODBS-Anbietern
  - Standardisierungsaktivitäten: ODMG und SQL3
- Schwächen von DB-Programmiersprachen (persistente Programmiersprachen)
  - unzureichende Mehrsprachenfähigkeit
  - ein allgemeines Sichtenkonzept ist nicht verfügbar
  - z.T. unzureichende mengenorientierte Anfragemöglichkeiten

3. OMG-Standardisierungen: UML und ODMG

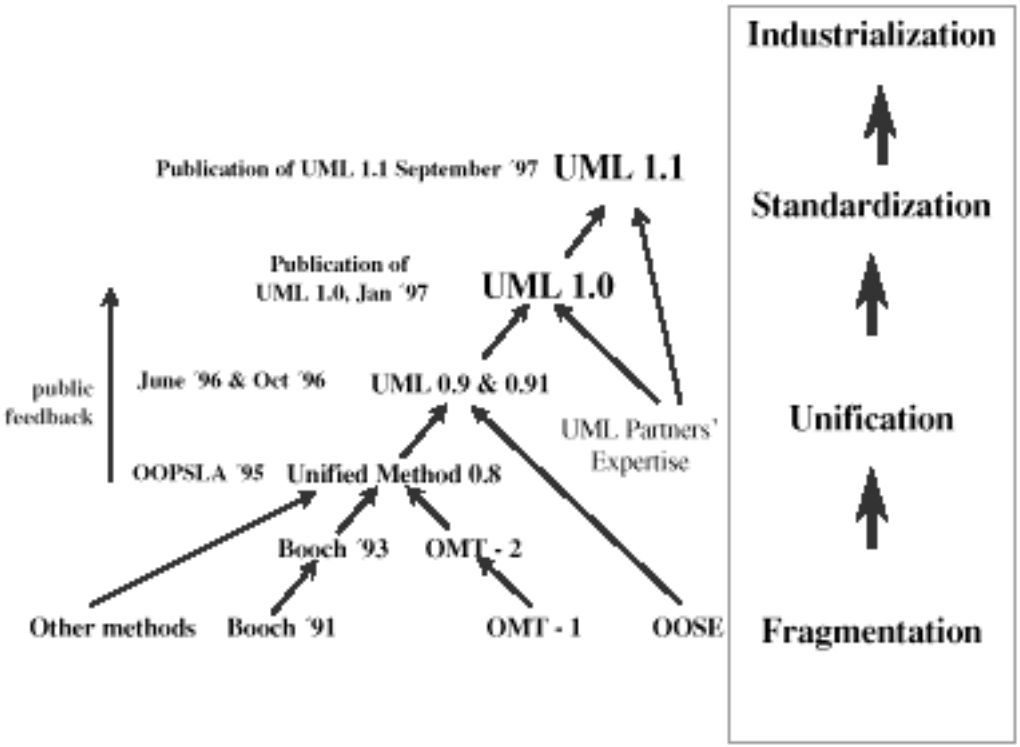
- OMG
- UML-Einblick
  - Einordnung
  - Klassendiagramme: Klassen/Objekte, Assoziationen, Aggregation/Komposition, Generalisierung
- ODMG-Objektmodell
  - Typen
  - einfache und strukturierte Objekte
  - Attribute und Beziehungen
- Objektdefinition (ODL)
- Anfragesprache (OQL)
  - Query-Definition und -Ausdrücke
  - Kollektionsausdrücke
  - sonstige Ausdrücke

Object Management Group (OMG)

- Herstellervereinigung zur Definition einer Gesamtarchitektur sowie von Standards zu verteilten objektorientierten Systemen
  - Gründung 1989
  - ca. 500 Mitgliederfirmen
- Ziel: umfassende Interoperabilität zwischen Objekten in heterogenen verteilten Systemen
  - bausteinartiges Zusammensetzen verteilter Anwendungen
  - Integration von neuen Systemen und Altsystemen (legacy systems)
- wesentliche Aspekte
  - Repräsentation von Daten und Funktionen durch gekapselte Objekte
  - transparenter Objektzugriff durch Client-Anwendungen
  - Kommunikation über Vermittlungskomponente (Object Request Broker)
- OMG-Spezifikationen
  - OMA (Object Management Architecture)
  - CORBA (Common Object Request Broker Architecture)
  - Kern-Objektmodell
- ODMG: OMG-Untergruppe für objektorientierte DBS
  - Gründung 1991
  - erster Standardisierungsvorschlag: ODMG93; 1997: ODMG 2.0

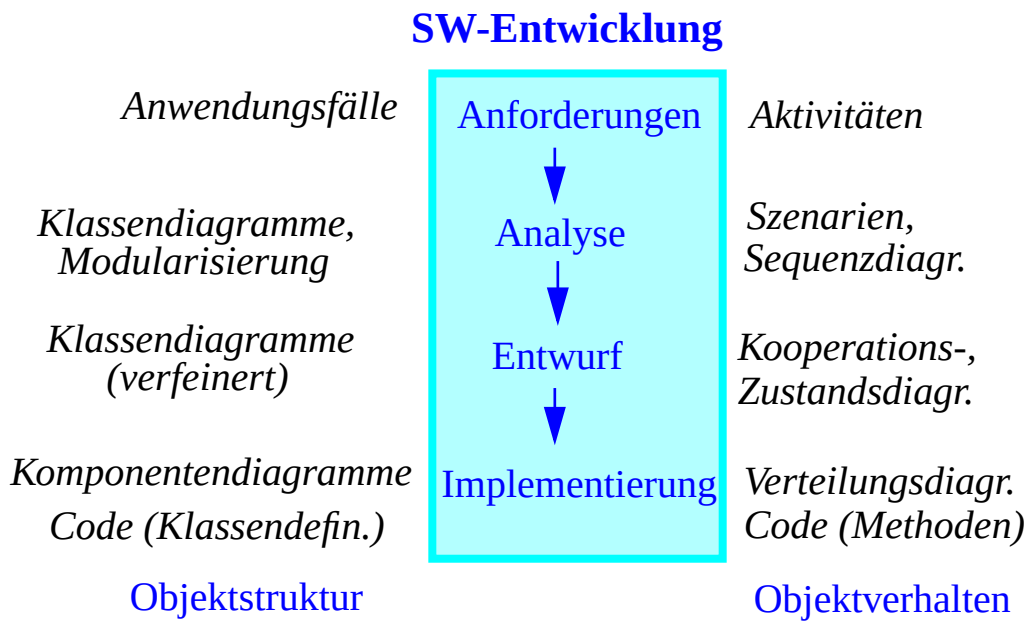
Unified Modeling Language (UML)

- standardisierte Notation/Sprache zur Beschreibung objektorientierter Software-Entwicklung
- Kombination unterschiedlicher Modelle bzw. Notationen:
  - Booch
  - Rumbaugh (OMT)
  - Jacobson (Use Cases)
- Standardisierung durch OMG (Object Management Group)



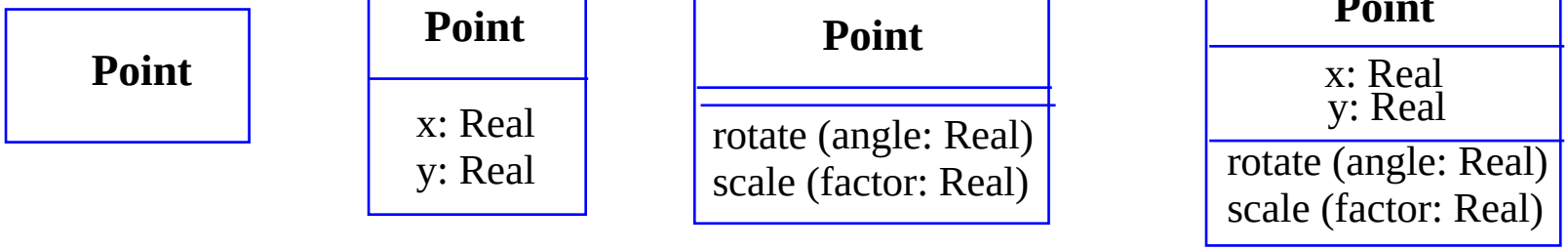
UML-Diagrammtypen

- Klassendiagramme
  - Klassen, Attribute und Operationen
  - Assoziationen
  - Aggregation und Komposition
  - Generalisierung
  - parametrisierte Klassen
- Paketdiagramme
- Anwendungsfälle (use case diagrams)
- Interaktionsdiagramme
  - Sequenzdiagramme (sequence diagrams)
  - Kollaborationsdiagramme (collabaration diagrams)
- Zustandsdiagramme (statechart diagrams)
- Aktivitätsdiagramme (activity diagrams)
- Verteilungsdiagramme (deployment diagrams)

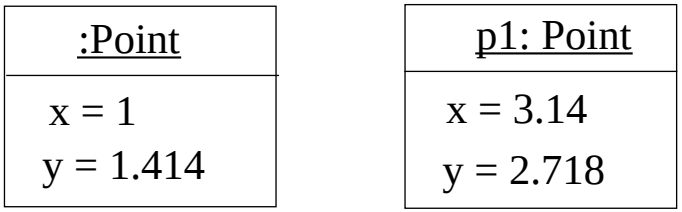


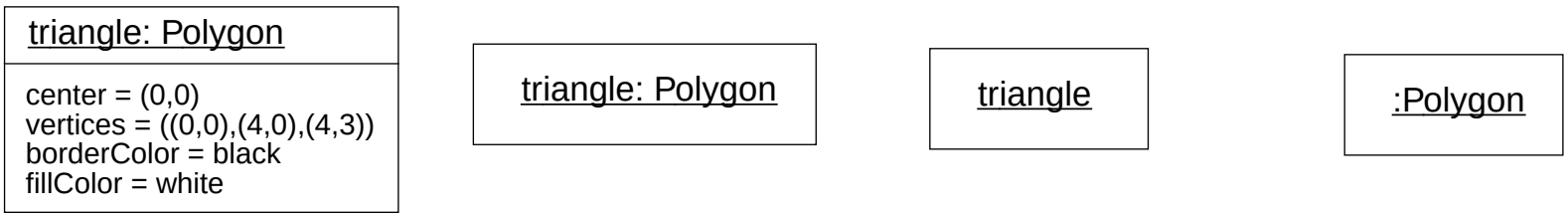
Darstellung von Klassen und Objekten

- Klassensymbol: Angabe von Klassenname, Attribute (optional), Methoden (optional)
  - i.a. werden nur relevante Details gezeigt



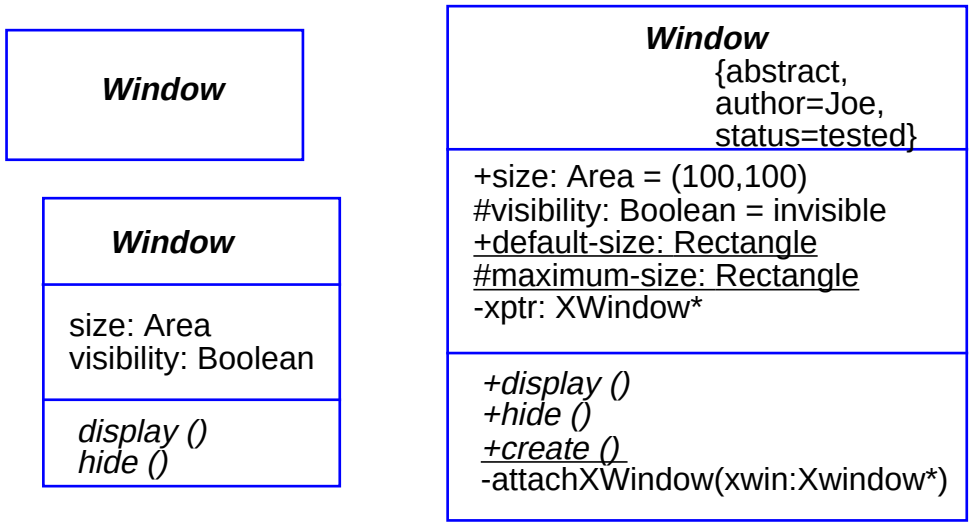
- analoge Darstellung von Klasseninstanzen (Objekten) -
  - keine Methodenangabe





Darstellung von Klassen (2)

- Detaildarstellung auf Implementierungsebene
  - Attributspezifikation: *Sichtbarkeit Name: Typ = Default-Wert { Eigenschaften }*
  - Operationen: *Sichtbarkeit Name (Parameterliste) : Rückgabeausdruck { Eigenschaften }*
  - Deklaration der Sichtbarkeit : öffentlich / public (+), geschützt / protected (#), privat (-)
  - unterstrichen: Klassen-Attribute / -Operationen



- Darstellung von Bedingungen (Constraints) innerhalb geschweifter Klammern **{ }**
  - Formulierung der Bedingungen in beliebiger Sprache möglich
  - OCL (Object Contraint Language) Teil der UML
- Tool-spezifische Erweiterungen möglich (Angabe von Exceptions, Verantwortlichkeiten, ...)

Assoziationen

- Repräsentation von Beziehungen (relationships)
- optional: Festlegung eines Assoziationsnamens, von Rollennamen sowie Kardinalitätsrestriktionen

- Kardinalitätsrestriktionen (multiplicity)

  kein Default-Wert (fehlende Angabe bedeutet “unspezifiziert”)

x..y	mindestens x, maximal y Objekte nehmen an der Beziehung teil
*	“viele”
0..*	optionale Teilnahme an der Beziehung
1..*	
0..1	
1	genau 1

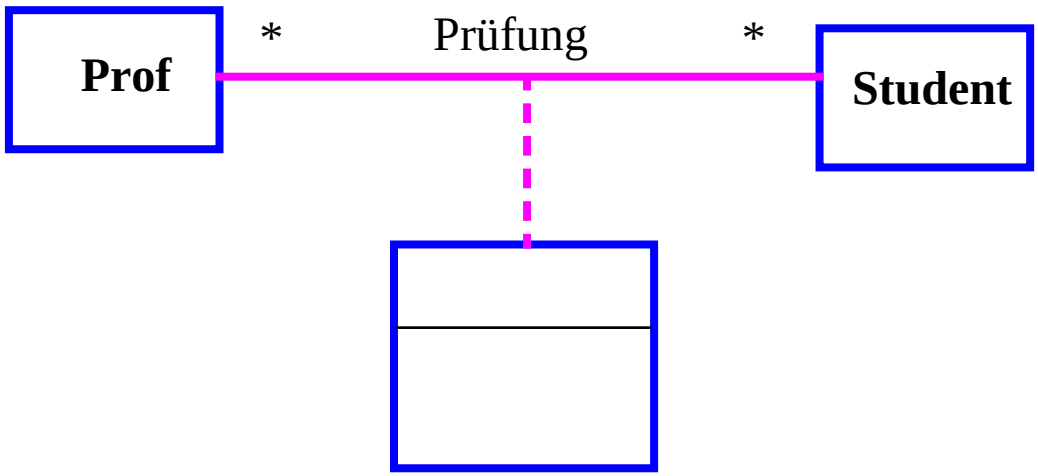
Assoziationen (2)

- **gerichtete Assoziation** (uni-directional association): Einschränkung der **Navigierbarkeit**
  - auf konzeptioneller Ebene nicht notwendigerweise festzulegen
  - regelt wo Verantwortlichkeit liegt, zugeordnete Objekte zu bestimmen
  - keine direkte Navigationsmöglichkeit in umgekehrter Richtung (einfachere Implementierung)

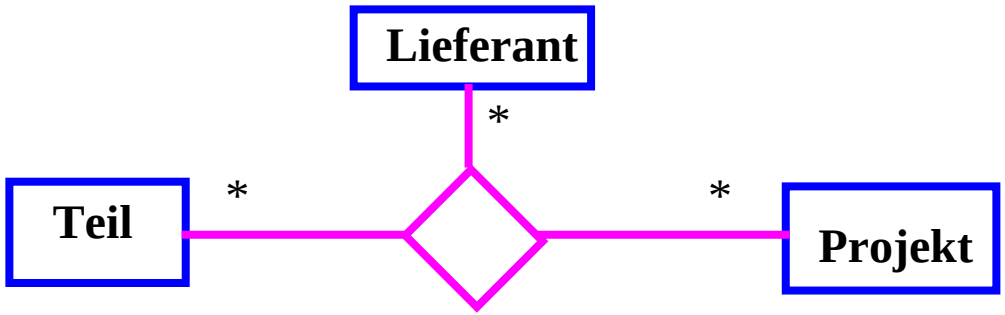
- für Beziehung kann Ordnung der beteiligten Objekte verlangt werden (List-Semantik): {ordered}
  - Default: {unordered}
- Festlegung der Sichtbarkeit für Rolle möglich (+, -, #)

Assoziationen (3)

- **Assoziations-Klassen:** notwendig für Beziehungen mit eigenen Attributen

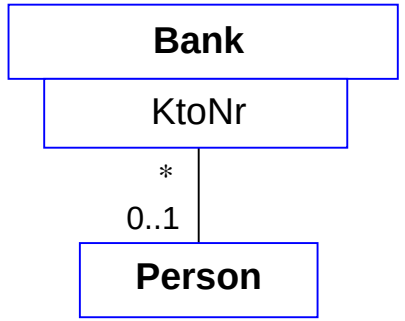


- 3-stellige Beziehung:

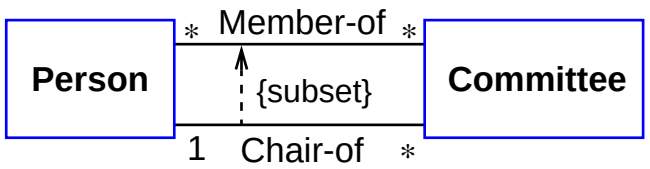


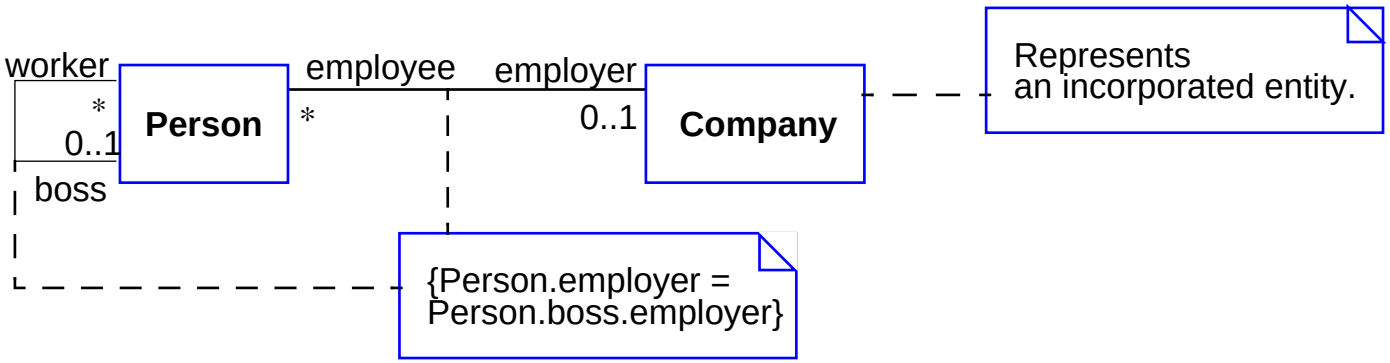
Assoziationen (4)

- **Qualifizierte Assoziation:** Festlegung von Attributen, welche für eine Assoziation eine Partitionierung der beteiligten Objekte bewirken
  - Partitionsattribut (e) gehört zur Assoziation
  - identifiziert zugehörige Objekte der an der Beziehung beteiligten Klasse



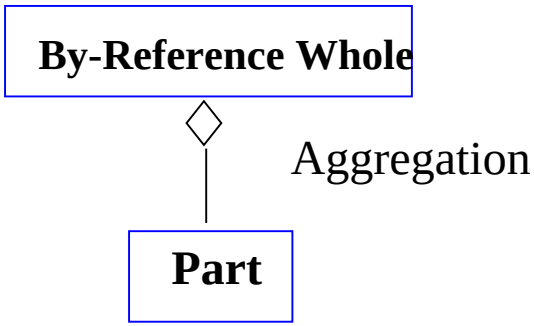
- Beispiele zur Verwendung von Constraints / Kommentaren

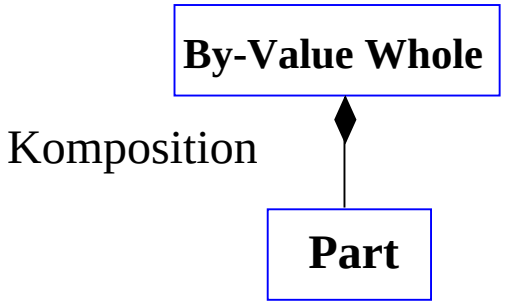




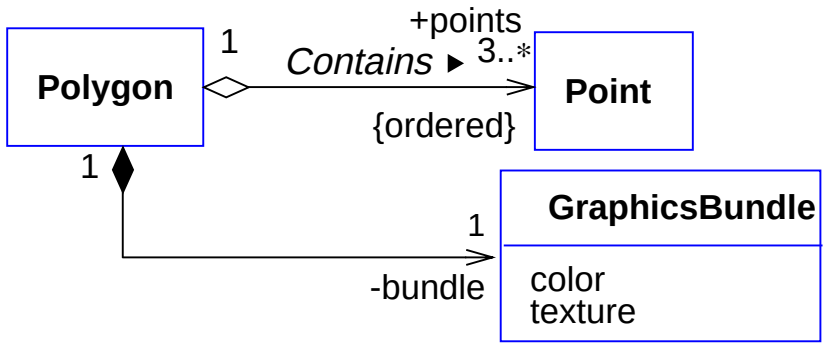
Aggregation

- **Aggregation**: spezielle Assoziation zwischen 2 Klassen, in der eine Klasse eine andere enthält



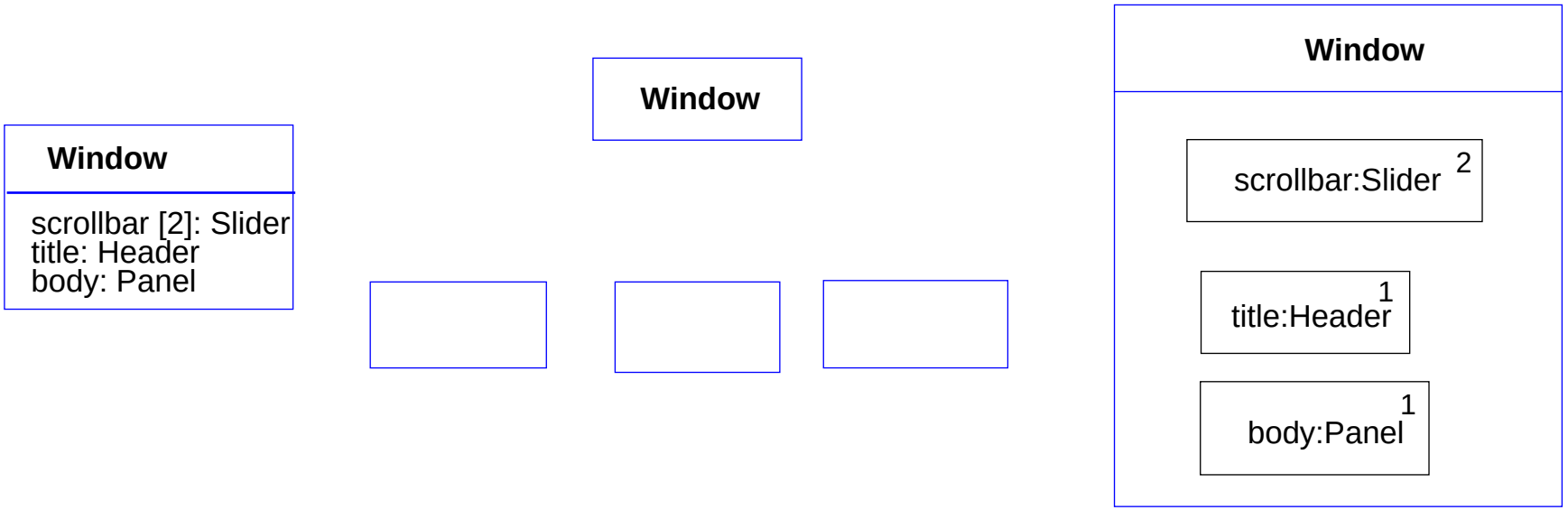


- **Komposition**: Aggregation, bei der eine Klasse ein Attribut einer anderen ist
  - Teil-Objekt gehört genau zu einem Aggregat-Objekt; Existenzabhängigkeit



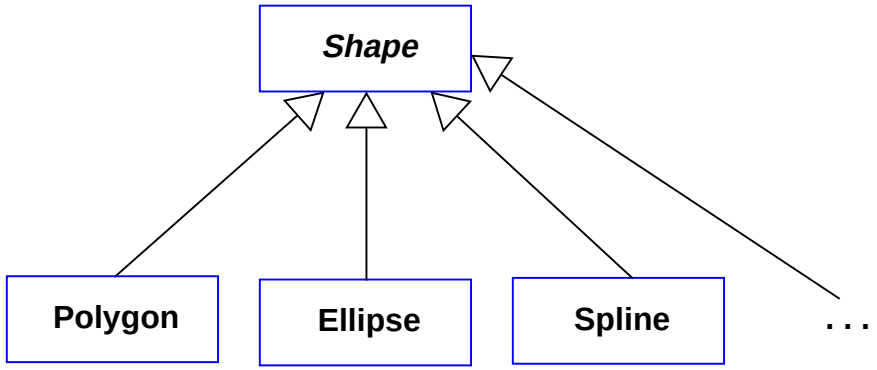
Aggregation (2)

- Unterschiedliche Darstellungsarten zur Komposition

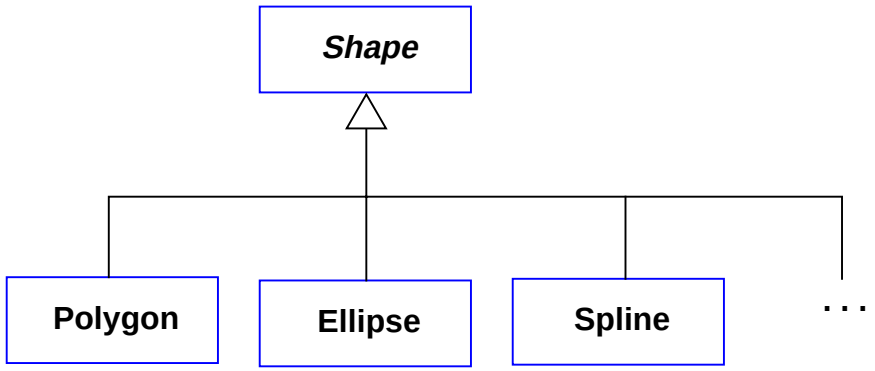


Generalisierung / Vererbung

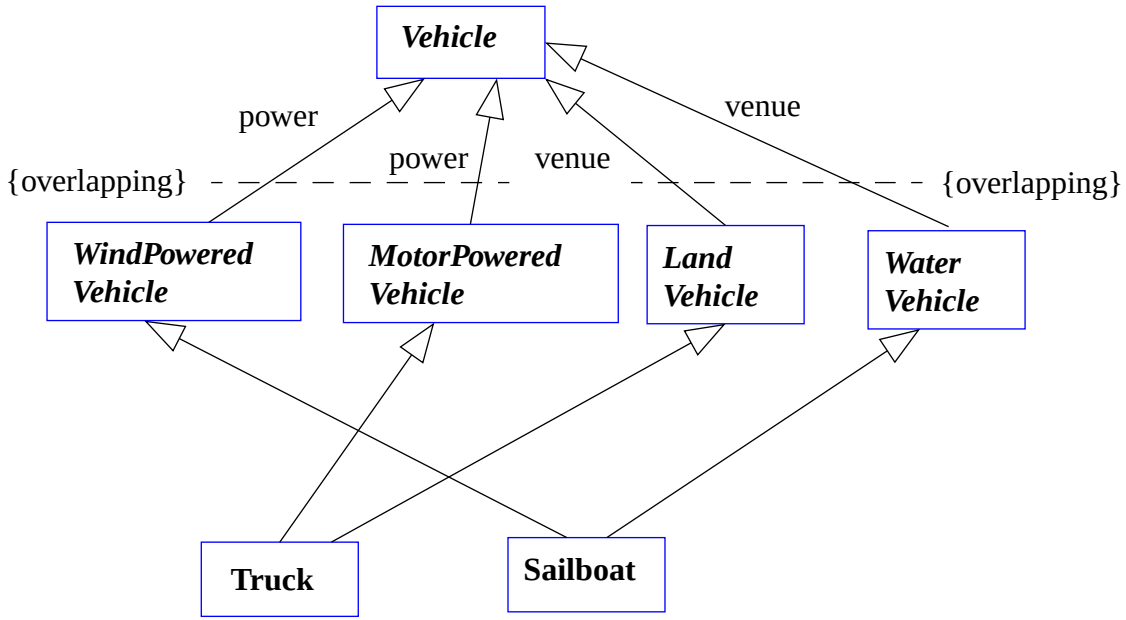
Separate Target Style

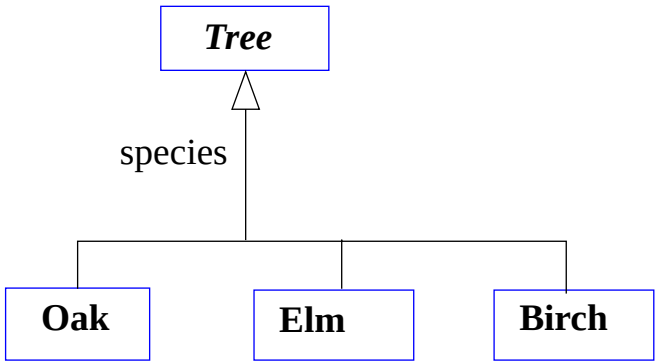


Shared Target Style

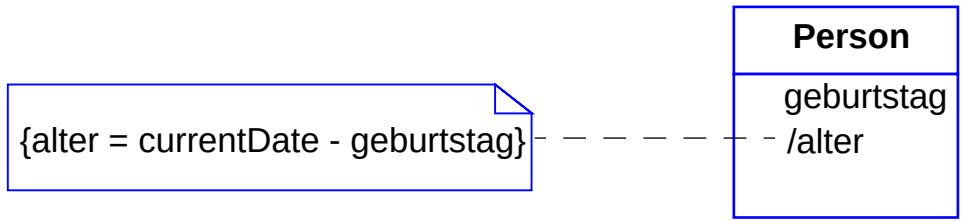


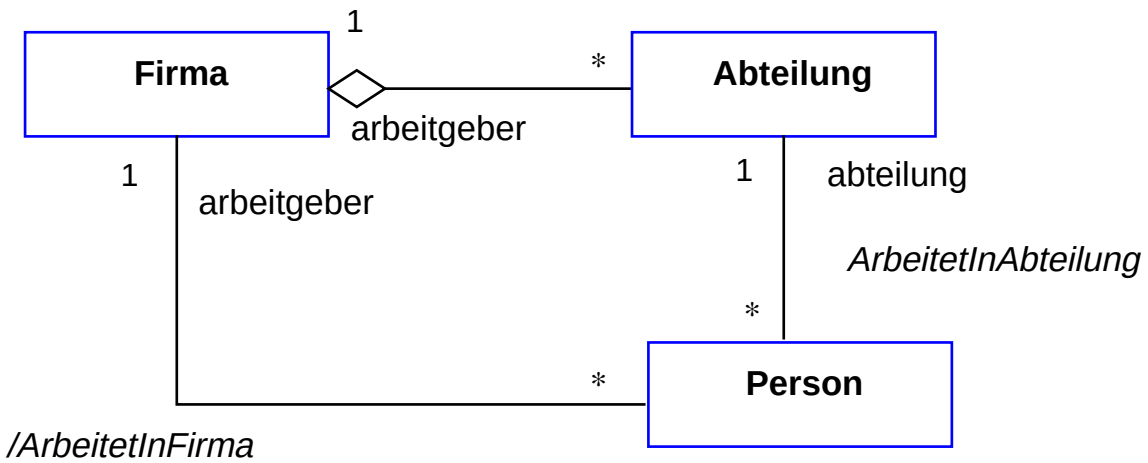
- Angabe von Diskriminatoren sowie Spezialierungsart (overlapping/disjoint, incomplete/complete)





Abgeleitete Attribute / Assoziationen

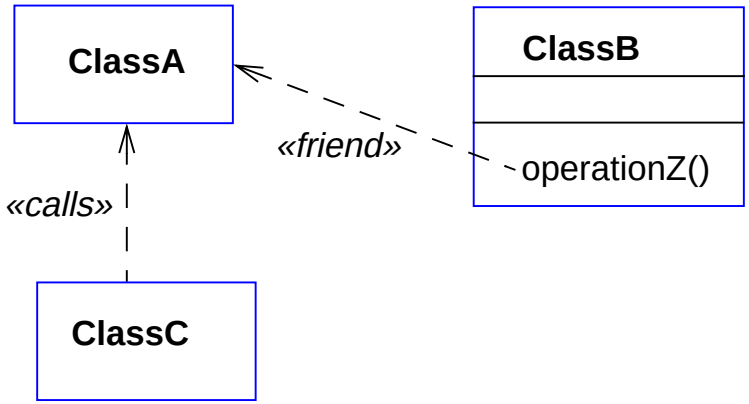




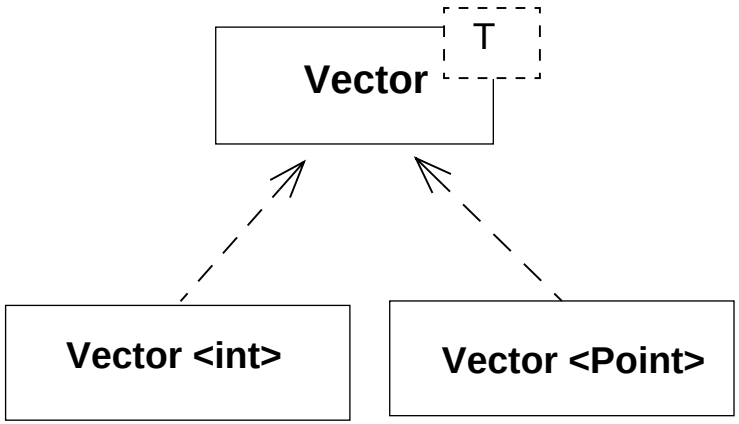
{ Person.arbeitgeber = Person.abteilung.arbeitgeber }

Abhängigkeiten (dependencies)

- Abhängigkeit liegt vor, falls Änderungen an der Definition eines Elementes Änderungen an der Definition eines anderen Elementes bedingen können
  - Darstellung durch gestrichelte Pfeillinien
  - Beispiele: uses, calls, friend, imports, ...



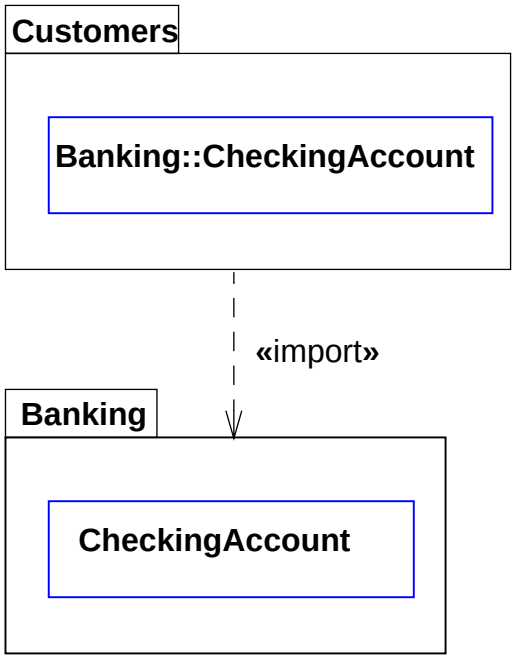
Parametrisierte Klassen

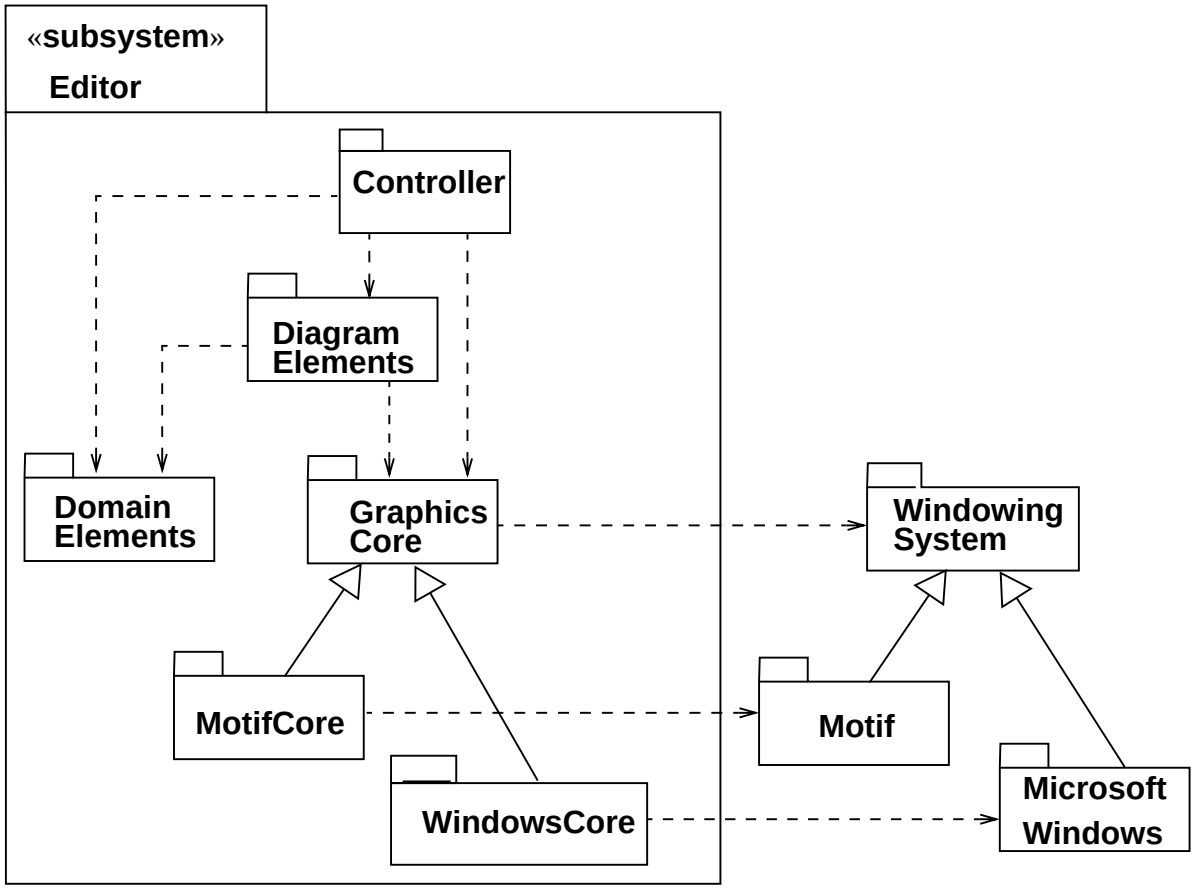


- Typparameter erlaubt Spezifikation von Klassenfamilien
- besonders vorteilhaft bei Kollektionsklassen / -typen (beliebige Komponententypen)

Paketdiagramme

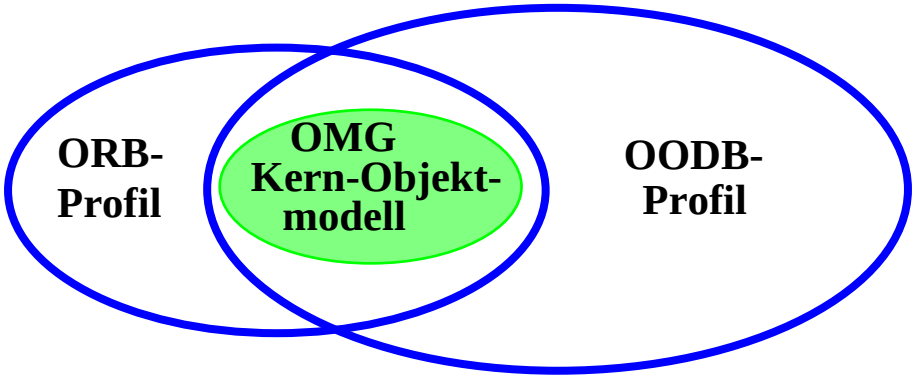
- Modellelemente können innerhalb von Paketen zusammengefaßt werden (-> Modularisierung)
  - Paketdiagramme veranschaulichen Abhängigkeiten zwischen Moduln





OMG-Objektmodelle

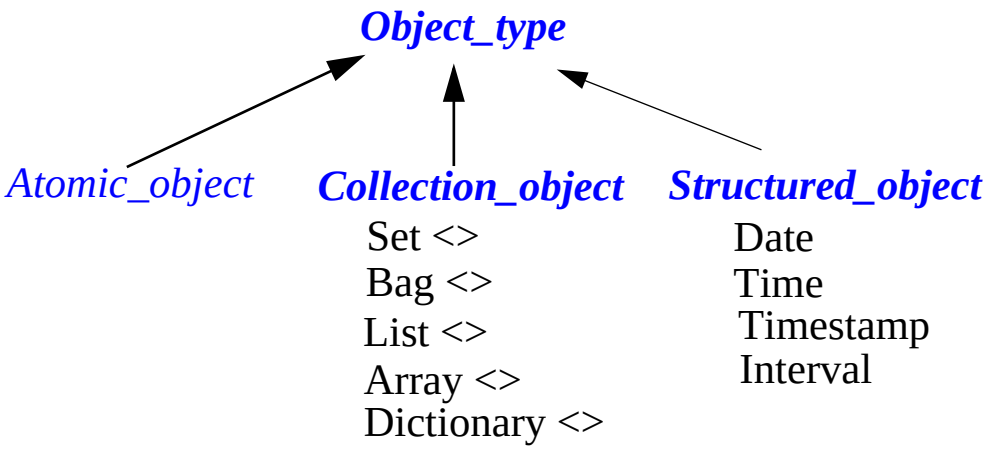
- OMG-Kernmodell: Identität, Typisierung, Operationen, Typhierarchien, Vererbung
- Profile: Kernmodell + kompatible Erweiterungen

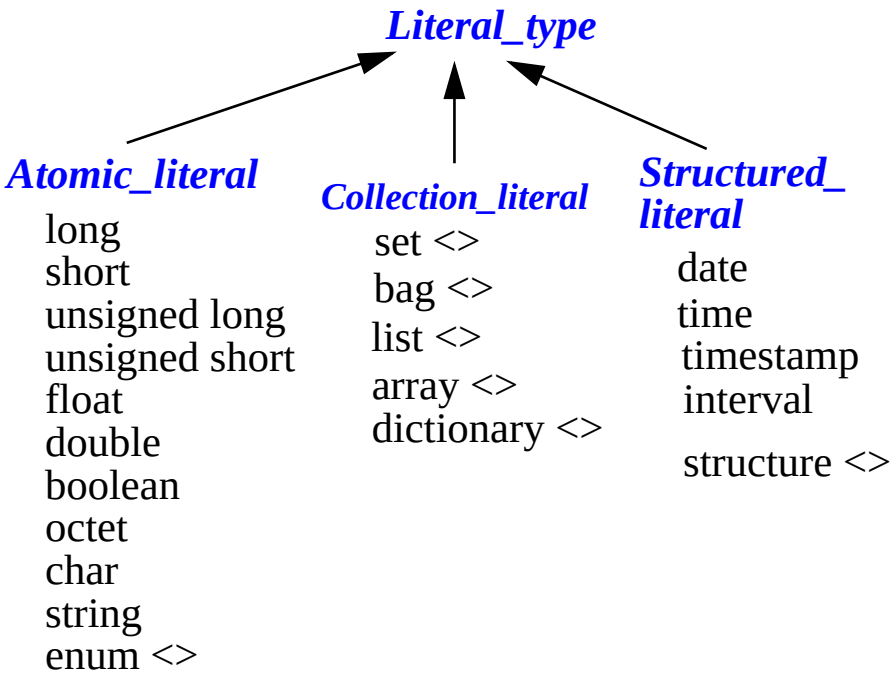


- OODB-Erweiterungen: Attribute, Beziehungen, Persistenz, Anfragen, Transaktionen
- ODMG-Standardisierung besteht aus
  - Objektmodell
  - Object-Definitionssprache (ODL)
  - Objekt-Anfragesprache (OQL)
  - Sprachbindungen (language bindings) für C++, Smalltalk und Java

ODMG-Objektmodell: Überblick*

- Unterscheidung zwischen Objekten (haben OID) und Literalen (haben keinenObjekt-Identifier)
- sowohl Literale als auch Objekte haben einen Typ





kursive Typen: nicht direkt instantiierbar !

- Objekte eines Typs weisen gemeinsame Charakteristika bezüglich Zustand und Verhalten auf
  - Zustand: Werte für bestimmte Eigenschaften (properties)
    a) Attribute
    b) (symmetrische) Beziehungen zu anderen Objekten
  - Verhalten: Menge von Operationen

* Cattell, R.: *Object Database Standard: ODMG 2.0*, Morgan Kaufmann, 1997.

Typen

- Typen bestehen aus
  - externer Spezifikation: Festlegung der sichtbaren Eigenschaften, Operationen und Ausnahmen (exceptions)
  - einer (oder mehreren) Implementierung(en): sprachabhängige Festlegung der Datenstrukturen und Methoden zur Realisierung der Schnittstelle

- Typ-Spezifikation ist abstrakt und implementierungsunabhängig; Beschreibung durch ODL
  - Verwendung von *Interface*-Definitionen (nur Operationen) oder *Klassen*-Definitionen (Operationen + Eigenschaften)
  - Klassen beschreiben direkt instanziierbare Typen, während Interfaces nicht direkt instanziierbarbar sind
  - *Literal*-Definition definiert abstrakten Zustand eines Literal-Typs

```
interface Employee { ... };
class Person { ... };
struct Complex {float re; float im; };
```

- Typ-Implementierung ist sprachabhängig und im DB-Schema nicht sichtbar
  - Festlegung der Repräsentation (Instanzvariablen) und Methoden
  - Kapselung
  - Mehrsprachenfähigkeit durch Language Bindings

Typen (2)

- Typen können durch Sub-Typen spezialisiert werden (IS-A-Beziehung, Vererbung von Operationen)
  - Definition zusätzlicher Operationen
  - Operationen können überladen werden (Overloading)
  - Einfach- und Mehrfachvererbung
  - Interfaces können von anderen Interfaces erben/abgeleitet werden, jedoch nicht von Klassen
  - Klassen können von Interfaces abgeleitet werden

- zusätzliche Zustands-Vererbung für Klassen: EXTENDS-Beziehung
  - nur Einfachvererbung
  - Unterklasse erbt alle Eigenschaften der Oberklasse

Typen (3)

- **Extent** eines Typs T: Menge aller Instanzen von T innerhalb einer bestimmten Datenbank
  - für Subtyp T2 von T gilt, daß der Extent von T2 eine Teilmenge des Extents von T ist
  - DB-Entwerfer kann entscheiden, für welche (Sub-) Typen explizite Extents angelegt werden
- Festlegung von Schlüsselkandidaten (Keys) möglich

```
class Vorlesung
    //          type properties:
        (       extent Vorlesungen;
                key (Vname, Semester); )
    //          instance properties:
        {       attribute string Vname;
                attribute string Semester;
                attribute string Dozent;
        //       instance operations:
                • • •
        };
```

Literale

- 11 atomare Literal-Typen (long, short ...) analog zu OMG IDL
- Kollektions-Literale
  - set, bag, list, array, dictionary
  - haben keine OID
  - Elemente können Literale oder Objekte sein!
- Strukturierte Literale
  - feste Anzahl i.a. heterogener und benamter Elemente (Literal oder Objekt)
  - zahlreiche Operationen für vordefinierte Strukturen (date, time, interval, timestamp)
  - Beispiel einer benutzerdefinierten Struktur:

```
class Person {
        struct Adresse {string Straße, short Hausnr, string PLZ, string Ort};
        attribute string Name;
        attribute Adresse Anschrift;
        }
```

- beliebige Kombinierbarkeit von Strukturen mit anderen Strukturen und Kollektionstypen

Objekte

```
interface ObjectFactory {
        Object new ();
        }
```

```
interface Object {
        enum Lock_Type {read, write, upgrade};
        exception LockNotGranted{};
        void lock (in Lock_Type mode) raises (LockNotGranted);
        boolean try_lock (in Lock_Type mode);
        boolean same_as (in Object anObject);   // Identitätstest
        Object copy ();
        void delete ();
        }
```

- explizite Erzeugung neuer Objekte durch new-Konstruktor
- Objekt-Identifier:
  - nur für Objekte (-> Referenzierbarkeit)
  - eindeutig innerhalb einer Datenbank
- Objekte können optional über Namen angesprochen werden
  - Namensvergabe i.a. für “Einstiegspunkte” in die Datenbank
  - flache Namensstruktur
- Objektlebensdauer: transient oder persistent
  - einheitliche Verwaltung beider Objektarten <-> relationale DBS

Kollektionsobjekte

- Kollektionen
  - Gruppierungen homogener Objekte (gleicher Typ)
  - Kollektionstypen sind *generische/parametrisierte Typen* (statische Festlegung der Elementtypen)
  - jede Kollektion ist ein Objekt (eigene OID)

- Kollektionstypen
  - Set <t>: keine Duplikate (Duplikate möglich bei Bag, List, Array)
  - List <t>: Ordnung wird beim Einfügen festgelegt
  - Arrays <t>: 1-dimensional, feste (jedoch änderbare) Länge
  - Dictionary <t,v>: ungeordnete Folge von Schlüssel/Werte-Paaren. keine Duplikate

- Cursorbasierter (navigierender) Zugriff auf Kollektionen über *Iteratoren*
  - Verwaltung der aktuellen Position
  - Operationen: *reset, next_position, get_element*
  - es können mehrere Iteratoren pro Kollektion definiert werden
  - Bidirektional Iteratoren: geordnete Kollektionen können vorwärts und rückwärts bearbeitet werden
  - stabile Iteratoren: Änderungen während der Iterierung bleiben ohne Auswirkungen

Kollektionsobjekte: Schnittstellen

```
interface Collection: Object {
   unsigned long cardinality();
   boolean    is_empty();
   void       insert_element (in any element);
   void       remove_element (in any element);
   boolean    contains_element (in any element);
   Iterator   create_iterator (in boolean stability);
   }
```

```
interface Set: Collection {
   Set        union_with (in Set other);
   Set        intersection_with (in Set other);
   Set        difference_with (in Set other);
   boolean    is_subset_of (in Set other);
   boolean    is_superset_of (in Set other);
   }
```

```
interface Bag: Collection {
   Bag        union_with (in Bag other);
   Bag        intersection_with (in Bag other);
   Bag        difference_with (in Bag other);
   }
```

```
interface Dictionary: Collection {
   exception KeyNotFound {any key};
   any lookup (in any key) raises (KeyNotFound);
   •••}
```

```
interface Iterator: Object {
   exception NoMoreElements{};
   boolean    is_stable();
   boolean    at_end();
   void       reset();
   any        get_element () raises (NoMoreElements);
   any        next_position () raises (NoMoreElements);
   }
```

```
interface List: Collection {
   void    replace_element_at (in unsigned long index,
                               in any element);
   void    remove_element_at (in unsigned long index);
   void    retrieve_element_at (in unsigned long index);
   void    insert_element_first (in any obj);
   void    insert_element_last (in any obj);
    • • •
   List    concat (in List other);
   }
```

```
interface Array: Collection {
   void    replace_element_at (in unsigned long index,
                               in any element);
   void    remove_element_at (in unsigned long index);
   void    retrieve_element_at (in unsigned long index);
   void    resize (in unsigned long new_size);
   }
```

Objekteigenschaften: Attribute und Relationships

- Attribute
  - sind jeweils genau 1 Objekttyp zugeordnet
  - Attributwerte sind Literale oder Objekt-Identifier (Nullwert: nil)
  - Attribute selbst sind keine Objekte (keine OID)

```
class Person {
        attribute string Name;
        attribute date Geburtsdatum;
        attribute enum Geschlecht {m, w};
        attribute Adresse Heimanschrift;
        atrribute set<string> Hobbies;
        attribute Abt Abteilung;
    }
```

Relationships

- Relationship
  - Beziehung zwischen 2 Objekttypen (nur binäre Beziehungen möglich)
  - 3 Arten: 1:1, 1:n, n:m
  - symmetrische Definition in beteiligten Objekttypen
  - jede Zugriffsrichtung (traversal path) bekommt eigenen Namen

```
class Vorlesung { • • •
        relationship set<Student> Hörer inverse Student::hört;
}
class Student { • • •
        relationship set<Vorlesung> hört inverse Vorlesung::Hörer;
}
```

- symmetrische Beziehungen erlauben automatische Wartung der referentiellen Integrität
- Repräsentation von n:m-Beziehungen ohne künstliche Objekttypen
- Pfade auf Objektmengen können geordnet sein (z.B. über List)
- Built-In-Operationen auf Traversierungspfaden:
  - *form/drop:* Einrichten/Löschen einer Mitgliedschaft eines einzelnen Objektes in 1:1- bzw. 1:n-Beziehung
  - Kollektionsoperationen für mehrwertige Beziehungen

Relationships: Beispiele

class **Fakultät**{
attribute string Fname;

}
class **Professor** {
attribute string Pname;

}
class **Student** {
attribute string Sname;

}

- 1:1-Beziehung (z.B. Fakultät - Dekan)
- 1:n-Beziehung (z.B. Fakultät - Student)
- n:m-Beziehung (z.B. Fakultät - Hiwi; Prüfung)

Operationen auf Datenbanken und Transaktionen

```
interface TransactionFactory {
        Transaction new ();
        Transaction current();
        }
interface DatabaseFactory {
        Database new ();
        }
interface Database{
   void     open (in string dbname);
   void     close ();
   void     bind (in any an_object,
                  in string name);
   any      lookup (in string object_name);
}
```

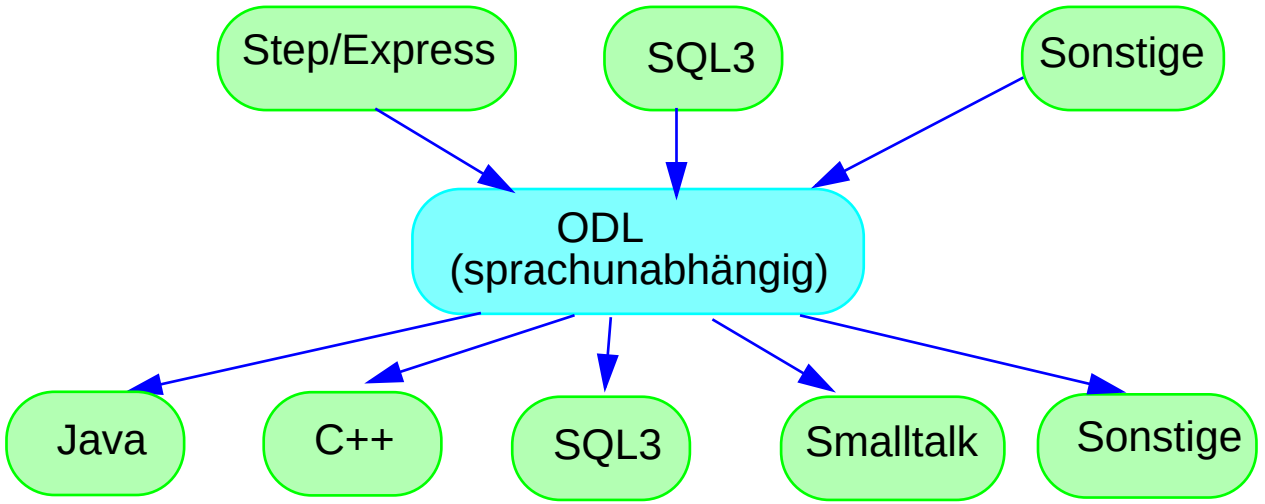
```
interface Transaction {
   exception TransactionInProgress{};
   exception TransactionNotInProgress{};
   void     begin () raises (TransactionInProgress);
   void     commit() raises (TransactionNotInProgress);
   void     abort() raises (TransactionNotInProgress);
   void     checkpoint() raises (TransactionNotInProgress);
   void     join():
   void     leave();
   boolean  isOpen();
}
```

- im Gegensatz zu ODMG-93 wird kein geschachteltes Transaktionskonzept mehr vorausgesetzt
  - 1 aktuelle Transaktion pro Thread; ggf. mehrere offene Transaktionen pro Thread (Wechsel mit leave / join)
  - *checkpoint*: entspricht *commit* mit nachfolgendem *begin*, jedoch ohne Freigabe der Sperren
- implizites oder explizites Locking auf Objekten
  - Read-, Write-, Upgrade- Locks
  - Upgrade-Locks explizit anzufordern (Vermeidung von Konversions-Deadlocks)
  - alle Sperren werden bis zum Transaktionsende gehalten (-> Serialisierbarkeit)
- Ausnahme-/Fehlerbehandlung über Exceptions

ODL (Object Definition Language)

- dient lediglich zur portablen Spezifikation von Interface-Beschreibungen von Objekttypen
  - keine vollständige Programmiersprache
  - Erweiterung der OMG Interface Definition Language (IDL)

- ODL kann auf verschiedene konkrete Sprachen abgebildet werden => unterstützt Zusammenarbeit zwischen heterogenen Systemen



  - z.Zt. Abbildung auf C++, Smalltalk und Java festgelegt

- neben ODL existiert noch Object Interchange Format (OIF) zum Datenaustausch

ODL (2)

- Typspezifikation durch Interface- oder Class-Deklaration

```
interface       ::=   interface identifier [: supertype_list]
                      {
                      [ interface_body ]
                      };
class           ::=   class identifier [ extends superclass_name]  [: supertype_list]
                       [ type_property_list ]
                      {
                            [ interface_body ]
                      };
type_property_list  ::=        ( [ extent extent_name ] [ key[s] key_list ] )
key                 ::=        property_name | ( property_name_list)
```

- Beispiel

```
class Professor: Person
                ( extent Profs;
                  keys PNR, (Name, Gebdat) )
                {
                        [ <interface_body> ]
                };
```

ODL (3)

- Instanzen-Properties: Attribute und Relationships

interface_body	::=	export | export interface_body
export	::=	type_dcl | const_dcl | except_dcl | attr_dcl | rel_dcl | op_dcl
type_dcl	::=	**typedef** type_spec declarators | struct_type | union_type | enum_type
struct_type	::=	**struct** identifier **{** member_list **}**
const_dcl	::=	**const** const_type identifier **=** const_exp
except_dcl	::=	exception identifier **{** [member_list] **}**
attr_dcl	::=	[**readonly**] **attribute** domain_type attr_name **[** fixed_array_size **]**
domain_type	::=	simple_type_spec | struct_type | enum_type
rel_dcl	::=	**relationship** target_of_path identifier **inverse** inverse_path [**{ order_by** attribute_list **}**]
target_of_path	::=	identifier | rel_collection_type < identifier >
inverse_path	::=	identifier **::** identifier

- Beispiel

```
class Professor: Person ( extent Profs;   keys PNR, (Name, Gebdat) )
{       struct Kind {string Kname, unsigned short Alter};
        attribute string Name;
        attribute unsigned long PNR [8];
        attribute date Gebdat;
        attribute set<Kind> Kinder;
        relationship ABT Abteilung inverse ABT::Abt_Angehörige
        relationship set<Student> Prüflinge inverse Student::Prüfer;
        ...
}
```

ODL (4)

- Definition der Operationen wie in IDL

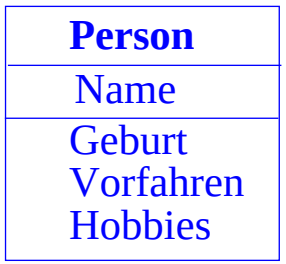
op_dcl	::=	[**oneway**] return_type identifier **(** [argument_list] **)** [exceptions_raised]
return_type	::=	simple_type_spec | **void**
argument	::=	role simple_type_spec identifier
role	::=	**in** | **out** | **inout**
exceptions_raised	::=	**raises** (exception_list)

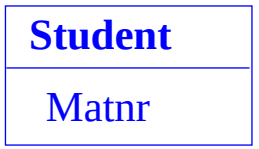
- Beispiele:

```
void Neuer_Pruefling (in Student s);

unsigned short Alter () raises (Geb_unbekannt);
```


ODL-Beispiel

```
class Person (extent PERS) {
        attribute string Name
        relationship set<Person> Elterninverse Person::Kinder;
        relationship  List<Person> Kinderinverse Person::Eltern;
        relationship  Apartment lebt_in inverse Apartment::wird_bewohnt_von;
        void Geburt ( in Person Kind);
        set< Person> Vorfahren ();
        set<string> Hobbies();
            };
class Angestellte: Person{
             attribute float Gehalt;
            };
class Student: Person{
             attribute string Matnr;
            };
class Gebäude{
             attribute struct Adresse {string Straße,string Ort};
             relationship  List<Apartment> Apartments inverse Apartment::Haus;
             Apartment preiswert ();
            };
class Apartment{
          attribute short Nummer;
          relationship Gebäude Hausinverse Gebäude::Apartments;
          relationship set <Person> wird_bewohnt_voninverse Person lebt_in;
            };
```





Spracheinbettung C++

- Realisierung der ODL in C++
  - ein zusätzliches Sprachkonstrukt für Beziehungen (neues Schlüsselwort **inverse**)
  - Klassenbibliothek mit Templates für Kollektionen etc.
  - Ref-Template Ref<T> für alle Klassen T, die persistent sein können

```
class Person {
    public:     String Name;
                Set  < Ref<Person> > Eltern
                        inverse Kinder;
                List < Ref<Person> > Kinder
                        inverse Eltern;
                Ref<Apartment> lebt_in
                        inverse wird_bewohnt_von;
// Methods:
    Person(); // Konstruktor
    void Geburt ( Ref<Person> Kind);
    Set< Ref<Person> > Vorfahren();
    virtual Set<String> Hobbies();
          };
class Angestellte: Person{
        public: float Gehalt;
};
class Student: Person{
        public: string Matnr;
       };
```

```
class Adresse{
        String Straße;
        String Ort;
       };
class Gebäude{
        Adresse adresse;
        List< < Ref<Apartment> > Apartments
                        inverse Haus;
        // Method
        Ref<Apartment> preiswert ();
};
class Apartment{
        int Nummer;
        Ref<Gebäude> Haus inverse Apartments;
        Set <Ref<Person>> wird_bewohnt_von
                        inverse lebt_in;
};

Set <Ref<Person>>  Pers;    // Personen-Extent
Set <Ref<Apartment>> Apartments;
```

OQL (Object Query Language)

- Spezifikation von Ad-Hoc-Anfragen sowie eingebetteter Queries

- Eigenschaften der OQL
  - keine vollständige DML bzw. OML
  - keine Änderungsoperationen (Änderungen müssen durch typspezifische Operationen erfolgen)
  - Verarbeitung beliebiger Objekter (nicht nur von Tupeln und Tabellen/Sets)
  - Unterstützung von Pfadausdrücken sowie von Methodenaufrufen innerhalb Queries
  - OQL-Aufrufe aus Anwendungsprogrammen möglich (für Programmiersprachen mit ODMG-Binding)
  - deklarativer Zugriff
  - an SQL angelehnt (insbesondere seit V2)

- funktionale Query-Sprache
  - Query = Ausdruck
  - orthogonale Verwendbarkeit von Query-Ausdrücken

- Beispiel:

```
select x.Matnr
from Studenten x
where x.Name="Schmidt"
```

OQL (2)

- Anfrageergebnis
  - ist entweder Kollektion von Objekten, Kollektion von Literalen, ein Objekt oder ein Literal
  - kann explizit strukturiert werden

```
select distinct x.Alter
 from Profs x
where x.Name = "Kurt"
```

```
select  struct (A: x.Alter, P: x.PNR)
 from Profs x
where x.Name = "Kurt"
```

```
select  struct (Prof: x.Name, sgS: (    select y
                                        from x.Prueflinge y
                                        where y.Note < 1,5))
 from Profs x
```

- *Iterator-Variablen:* äquivalente Notation e **as** x oder e x oder x **in** e
  (e sei Kollektionstyp über t, dann ist x vom Typ t)

OQL: Pfadausdrücke

- direkte Traversierung entlang von Relationships anstelle von Joins
  - Zugriff über "." oder "->" - Notation
  - Voraussetzung: keine der Referenzen weist Nullwert (nil) auf

    p.lebt-in.Haus.Adresse.Straße

- Zugriff bei mengenwertigen Beziehungen über *select*

- Pfad-Ausdrücke in From- und Where-Klausel möglich

- Berechnung von Joins kann weiterhin notwendig sein

- Abprüfen auf Nullwerte: *is_defined* bzw. *is_undefined*

OQL (4)

- Verwendung von Methodenaufrufen wie Attribut- oder Relationship-Zugriff möglich
  - kein syntaktischer Unterschied bei parameterlosen Methoden
  - Methode kann komplexen Wert liefern -> Verwendung innerhalb von Pfadausdrücken möglich

```
select max (select k.Alter from p.Kinder k)
from Pers p
where p.Name = "Paul"

select p.ältestes_Kind.Name
from Pers p
where p.lebt_in ("Leipzig")
```

- Benamte Query-Definition
  - allgemeine Form: **define** id (x1, ... xn) **as** e (x1, ...xn)
  - Beispiele:

```
define Schmidt () as
        select p from Pers p where p.Name= "Schmidt"

define Alter (x) as
        select p.Alter from Pers p where p.Name=x
```

- Mengenausdrücke (anwendbar auf Set- bzw. Bag-Objekte)

query :== query **intersect** query | query **union** query | query **except** query

OQL: Kollektionsausdrücke

- Syntax

  query ::= **for all** id **in** query: query | **exists** id **in** query: query | query **in** query |
  select [**distinct**] query **from** id **in** query {, id **in** query} [**where** query] [**order by** query {, query}|
  sort id **in** query **by** query {, query} |
  count (query) | **sum** (query) | **max** (query) | **min** (query) | **avg** (query) |
  select-query **group by** partion-attributes [**having** predicate]

- Bemerkungen / Beispiele
  - Ergebnis der Quantifikations- und Mitgliedschaftstests ist true oder false

    for all x in Studenten: x.Matnr > 0
  - Select liefert Set (distinct) bzw. Bag
  - Sortierbeispiel:

    sort x in Personen by x.Alter, x.Name
  - Aggregatfunktionen: Count, Sum, ...

OQL: Gruppierung

- Syntax: select-query **group by** partion-attributes [**having** predicate]

  - Zerlegung des Select-Ergebnisses in Partitionen gemäß der Partitionierungsattribute
  - für Partitionierungsattribute a_1: e_1, ... a_n: e_n ist das Ergebnis vom Typ
    set <struct (a_1: type_of (e1), ..., a_n: type-of(e_n), *partition:* bag<type_of (grouped elements) >)>
  - Having: Auswahl unter den Partitionen

- Beispiele

```
select *
from Pers p
group by    niedrig: Gehalt < 2000,
            mittel: Gehalt >= 2000 and Gehalt < 6000,
            hoch: Gehalt > 6000
```

```
select Abteilung, DGehalt: avg (select x.p.Gehalt from partition x)
from Pers p
group by Abteilung: p.Anr
having avg(select x.p.Gehalt from partition x) > 5000
```

OQL (7)

- Konstruktionsausdrücke
  - Ausdrücke der Form t (p1: e1, ..., pn: en) mit t=Typ-Name, p=Property-Name, e=expression zur Erzeugung von Objekten zu bestehenden Typen

    Bsp.: *Professor (Name: "Kurt", PNR: 4711, Abteilung: INF)*
  - Dynamische Erzeugung neuer Struktur- und Kollektionsinstanzen

    Bsp.: *Struct (Name: "Peter", Alter: 25);*
    Set (1, 2, 3)
    List (1, 2, 2, 3)
    Array (3, 4, 2, 1, 1)
- Zugriff auf Listen und Arrays (indexed collections)
  - Zugriff auf i-tes Element: e [i] (i=0 liefert erstes Element)
  - Bilden von Teilkollektionen: e [i:j] (i-tes bis j-tes Element)
  - Zugriff auf erstes bzw. letztes Element: *first (e), last (e)*
  - Konkatenation: e1 + e2
- Konversionsausdrücke
  - *element* (e): Extraktion eines Elementes aus einer Kollektion (erfordert genau 1 Element)

    element (select x from x in Profs where x.Name = "Rahm")

    first (element (select x from x in Profs
    where x.Name = "Rahm").Publikationen)
  - *listtoset*: Umwandlung einer Liste in einen Set
  - *flatten* (e): "Flachklopfen" mehrstufiger Kollektionen

Kombinierbarkeit von OQL-Ausdrücken: Beispiel

- Bestimme Name der Straße, in der Angestellte mit dem geringsten Durchschnittsgehalt wohnen

1. Extent Angestellten aus Extent Pers (Typ Person, Subtyp Angestellte) ableiten

   define Angestellten() as select (Angestellte) p from Pers p

2. Gruppierung von Angestellten nach Straßen und Bestimmung des Durchschnittsgehalt

   define Gehalts_Info() as
   select struct (Straße, DGehalt: avg (select x.a.Gehalt from partition x))
   from Angestellten() a
   group by Straße: a.Adresse.Straße

3. Sortierung über Durchschnittsgehalt

   define sortierte_Gehalts_Info() as select g from Gehalts_Info() g order by g.DGehalt

4. Ausgabe der gesuchten Straße

- Einzelne Anfrage

Zusammenfassung

- UML (Unified Modeling Language)
  - umfassender Standardisierungsansatz zur objektorientierten Modellierung und Software-Entwicklung
- ODMG-Objektmodell
  - Erweiterung des OMG-Kernmodells
  - Objekte und Literale; Attribute und Relationships; Kollektionstypen und Strukturen
- Schemabeschreibung über ODL
- Anfragesprache OQL
  - deskriptive Teilsprache
  - hohe Orthogonalität
  - Queries können Pfadausdrücke sowie Methodenaufrufe enthalten
  - Anpassung an SQL wurde verbessert
- Beschränkungen
  - nur binäre Beziehungen
  - unzureichende Mechanismen zur Definition von Integritätsbedingungen
  - kein Sichtkonzept
  - keine Änderungen über Query-Ausdrücke
  - keine Autorisierung

4. Beispielrealisierungen von OODBS

- NF^2
- GemStone
- O_2

NF^2-Modell

- **N**on-**F**irst **N**ormal **F**orm (’nested relations’, unnormalisierte Relationen)
- wie Relationenmodell, jedoch können Attributwerte auch Relationen (Mengen von Tupeln) sein
- Probleme bei netzwerkartigen Beziehungen
- Polyeder-Beispiel

```
CREATE  TABLE Volumen
        { [     VId             INT,
                Bez             CHAR(20),
                AnzFlaechen     INT,
                Flaechen        { [     FId             INT,
                                        AnzKanten       INT,
                                        Kanten  { [     KId             INT,
                                                        Punkte  { [     PId     INT,
                                                                        X       INT,
                                                                        Y       INT,
                                                                        Z       INT
                                                                ] }
                                                ] }
                                ] }
        ] }
```

{ } *Mengenkonstruktor*, **[ ]** *Tupelkonstruktur*

NF^2-Ausprägungen

Volumen							
VId	Bez	Flaechen					
		FId	Kanten				
			KId	Punkte			
				PId	X	Y	Z
0	Tetraeder	1	12	123	0	0	0
				124	100	0	0
			13	123	0	0	0
				134	50	44	75
			14	124	100	0	0
				134	50	44	75
		2	12	123	0	0	0
				124	100	0	0
			23	123	0	0	0
				234	50	87	0
			24	124	100	0	0
				234	50	87	0
		3	13	123	0	0	0
				134	50	44	75
			23	123	0	0	0
				234	50	87	0
			34	134	50	44	75
				234	50	87	0
		4	14	124	100	0	0
				134	50	44	75
			24	124	100	0	0
				234	50	87	0
			34	134	50	44	75
				234	50	87	0

NF^2-Modell (3)

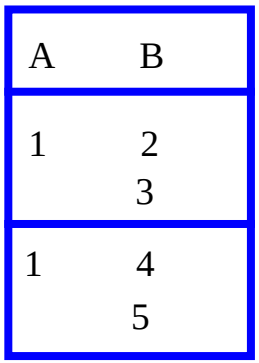
- Komplexes Objekt durch ein Tupel dargestellt:

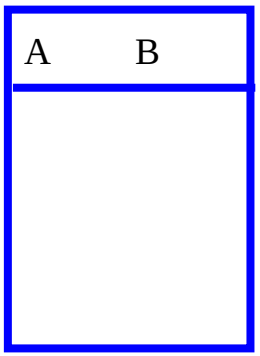
```
SELECT  *
FROM  Volumen
```

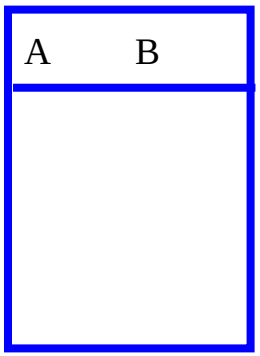
- Zugriff auf Teilkomponenten:

```
SELECT
      (SELECT
            (SELECT  X, Y, Z  FROM Punkte)
            FROM Flaechen
            WHERE FId=2 AND AnzKanten=3)
      FROM Volumen
WHERE VId=0
```

- Erweiterte relationale Algebra
  - Erweiterung von Projektion und Selektion auf geschachtelte Strukturen
  - ***NEST***-Operation: Erzeugen geschachtelter Relationenformate aus flachen Relationen
  - ***UNNEST***-Operation: Normalisierung (“Flachklopfen”) geschachtelter Relationen
  - NEST und UNNEST sind i.a. nicht invers zueinander !







Erweiterter natürlicher Join

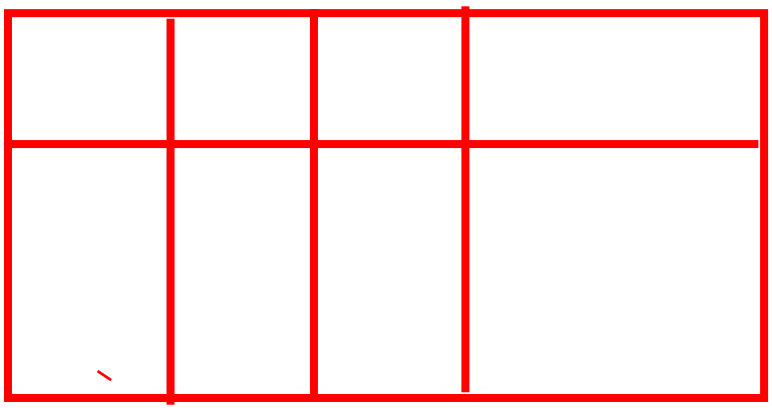
R1

A	B	X	
		C	D
a1	b1	c1	d1
		c2	d2
		c1	d3
a2	b2	c1	d2
		c3	d2

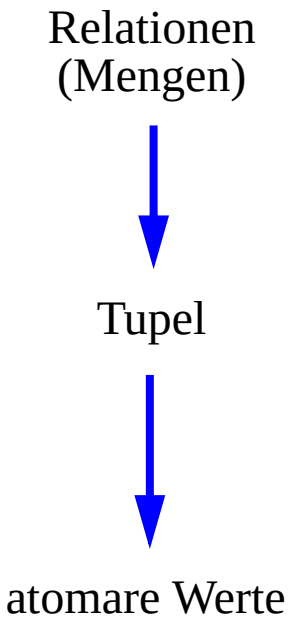
R2

E	B	X	
		C	D
e1	b1	c1	d1
		c1	d3
		c3	d4
e3	b2	c3	d2

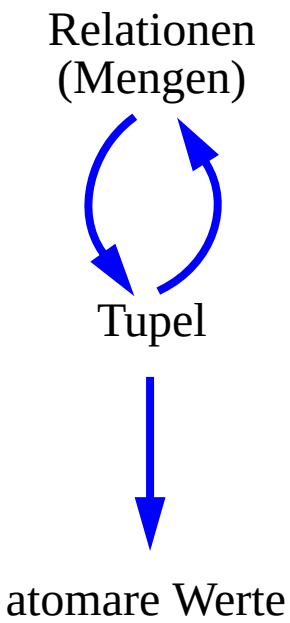
R1 ⋈ R2



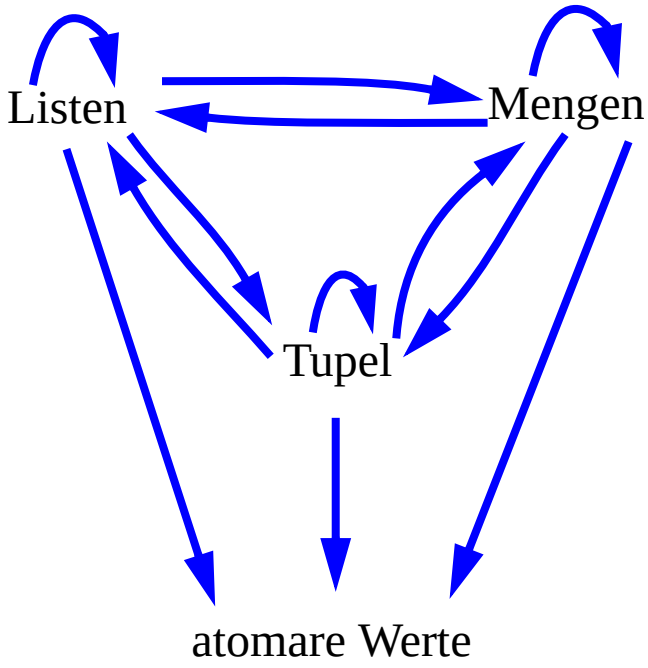
Anwendbarkeit von Typkonstruktoren



Relationenmodell



NF2-Modell
(DASDBS-Projekt)



Erweitertes NF2-Modell
(AIM-Projekt)

Bewertung des NF^2-Modells

- Vorteile:

  - einfaches Relationenmodell als Spezialfall enthalten
  - Unterstützung komplex-strukturierter Objekte
  - reduzierte Join-Häufigkeit
  - Clusterung einfach möglich
  - sicheres theoretisches Fundament (NF2-Algebra)

- Nachteile:

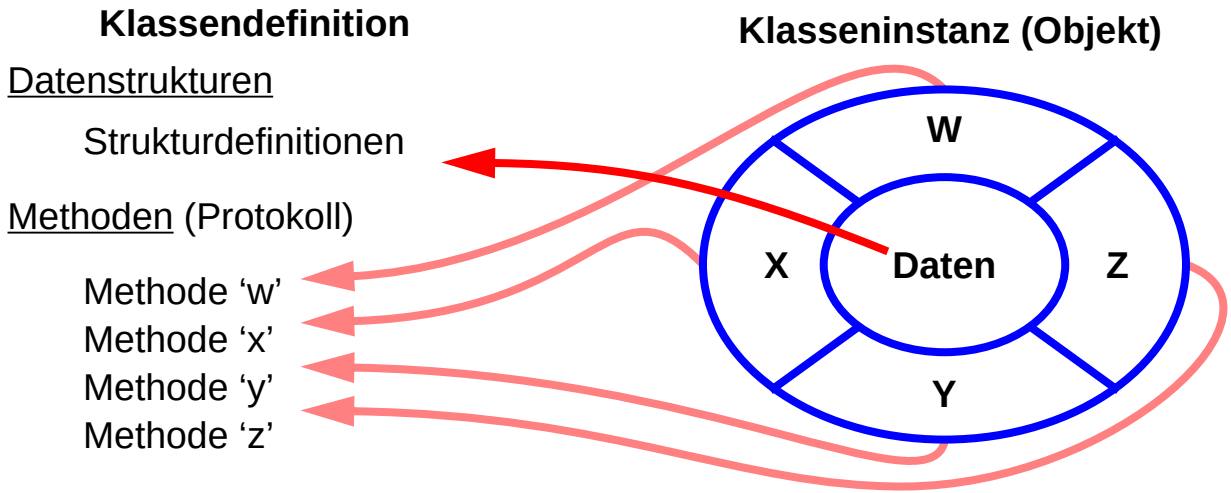
  - überlappende/gemeinsame Teilkomponenten führen zu Redundanz
  - unsymmetrischer Zugriff
  - rekursiv definierte Objekte nicht zulässig
  - keine Unterstützung von Generalisierung und Vererbung

GemStone

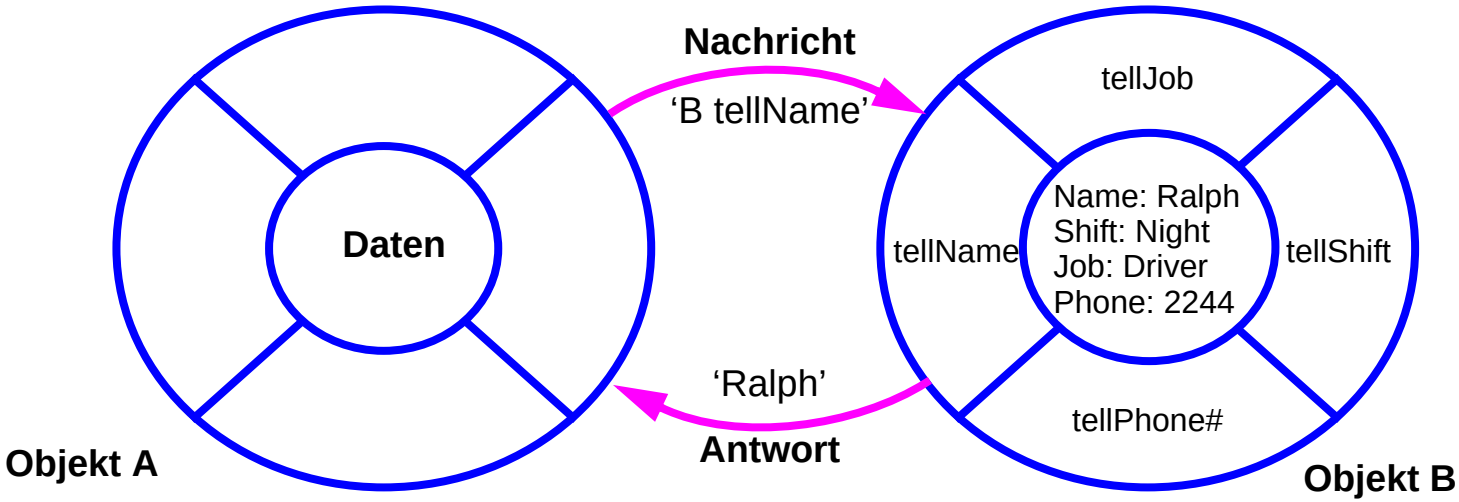
- OODBS-Produkt von ServioLogic Corp.
  - bereits seit Ende 1987 kommerziell verfügbar
  - basierend auf Smalltalk-80
- Sprache OPAL:
  - DDL
  - DML
  - allgemeine Programmiersprache
- Unterstützung von
  - Objekten bestehend aus:
    - internem Status beschrieben durch ***Instanzvariablen***
    - einer externen Schnittstelle (Menge von Methoden)
  - Nachrichten zum Aufruf von Methoden
  - Objektidentität (OOP = Object-oriented pointer)
  - Generalisierungshierarchie/Vererbung
  - Overloading, Late Binding
  - Workstation-/Server-Architektur

Objekte

- Definition von Objekten



- Objekte kommunizieren miteinander über Nachrichten



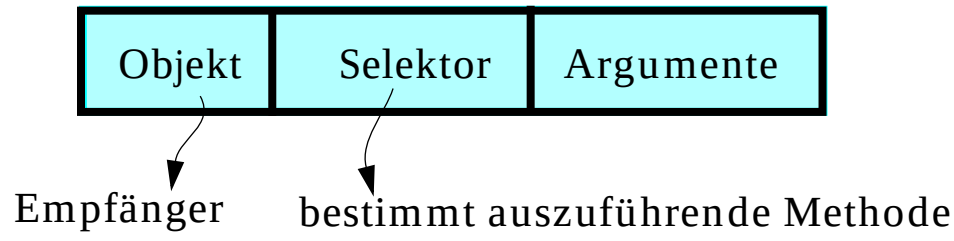
Klassenhierarchie von Gemstone (Auszug)

Object -
- Association - SymbolAssociation
- Behavior -
  - Class
  - MetaClass
- Boolean
- ClampSpecification
- Collection
- CompiledMethod
- Magnitude
- MethodContext - Block - SelectionBlock
- Segment
- Stream - PositionableStream
- System
- UndefinedObject
- UserProfile

Magnitude -
- Character
- DateTime
- Number -
  - Float
  - Fraction
  - Integer -
    - LargeNegativeInteger
    - LargePositiveInteger
    - SmallInteger

Collection -
- SequenceableCollection -
  - Array -
    - CollisionBucket
    - InvariantArray
    - Repository
    - SymbolList
  - String - InvariantString - Symbol
- Bag - Set -
  - Dictionary -
    - IdentityDictionary
    - SymbolDictionary - LanguageDictionary
  - SymbolSet
  - UserProfileSet
- HashDictionary -
  - IntegerHashDictionary
  - SymbolDictionary - LanguageDictionary

GemStone: Nachrichtenaufbau



- Sender bestimmt 'was' zu tun ist (Selektor), Empfänger das 'wie' (Abarbeitung der entsprechenden Methode)
- Nachricht bewirkt Zustandsänderung beim Empfänger oder/und Rückgabe einer Antwortnachricht
- ***Nachrichtentypen:***

a) einstellig (keine Argumente)

```
7  negated
P123  Vorname
sum sqrt
```

b) zweistellig (1 Argument, Selektor: 1 o. 2 Zeichen):

```
8 * 4
myObject = yourObject      (* Gleichheit *)
myObject == yourObject     (* Identität *)
(P1 Gehalt) <= (P2 Gehalt)
```

c) Schlüsselwort-Nachrichten

1 Empfänger, mehrere Keyword (Selektor)-/Argument-Paare

```
liste addFirst: elem
arrayOfStrings  at: (2+1) put:  'New York'
```

OPAL: Anwendungsbeispiel

- Erzeugen von Objekklassen

a) Tupelobjekte

```
Object  subclass: 'Person'
        instVarNames: #('Name', 'Abt', 'Gehalt')
        constraints:    # [ #[ #Name, String ],
                          #[ #Abt, String ],
                          #[ #Gehalt, Integer ] ]

Person  subclass:       'Programmierer'
        instVarNames: #('Sprache')
        constraints:    # [ #[ #Sprache, String ] ]
```

b) Mengenobjekt

```
Set     subclass:       'Pers-Set'
                        instVarNames:  #()
                        constraints: Person
```

- Ausführen von Operationen

```
PERS := Pers-Set new.       (* Erzeugen der 'Relation' PERS *)
PERS  insertName: 'Fred'  insertAbt:  'Vertrieb' insertGehalt: 45000 (*  Einfügen eines neuen Angestellten *)
```

Zugehörige Methoden-Definition:

```
method:  Pers-Set insertName:  na  insertAbt:  abt  insertGehalt: gehalt
         |  NewPers  |                (*  lokale Variable  *)
         NewPers := Person new.
         NewPers storeName: na; storeAbt: abt;  storeGehalt: gehalt.
         self  add:  NewPers.
%
```

O_2

- entwickelt im Rahmen des Forschungsprojektes ALTAIR (1986-1991)
- Projektpartner u.a.: INRIA, SNI, Universität Paris-Süd (F. Bancilhon et al.)
- seit 1991 Vertrieb durch *O_2-Technology* (in Deutschland durch *Nexus GmbH*)

- Client/Server-Architektur
- Verschiedene Sprachschnittstellen: C++, O_2C, C, Java (geplant)
- Unterstützt OQL
- Graphische Entwicklungstools und Bibliotheken (O2Tools, O2Kit, O2Look)

O_2: C++-Interface

- übernommene Eigenschaften von C++
  - Klassenattribute sind öffentlich (public), privat oder geschützt (protected)
  - Konstruktoren zur Objekterzeugung, Destruktoren zum Löschen
  - Einfach- und Mehrfachvererbung
  - Methoden und Operatoren können überladen werden
  - generische Klassen (Templates)

- Erweiterungen gegenüber C++
  - persistente Objekte
  - zusätzliche Templates für Kollektionen: d_Collection, d_Set, d_Bag, d_List, d_Array
  - Standard-Methoden auf Kollektionen (insert_element, remove_element, exists_element, ...)
  - Zugriff auf alle Elemente einer Kollektion: d_Iterator-Klasse (next, get_element, ...)
  - Unterstützung von inversen (bidirektionalen) Beziehungen zur Wahrung der referentiellen Integrität
  - Transaktionsverwaltung
  - Query-Zugriffsmöglichkeit (z.B. durch Angabe eines OQL-Prädikats als Parameter der Methode query)

O2: C++-Beispiel

```
class PERS   {public:      d_String Name;
                           int    Alter;
                           d_Ref<ABT> Abteilung inverse ABT::Angestellte;
                           //Konstruktor:
                           PERS (d_String n) { Name = n; };
              private:     int Gehalt;
             };

class ABT   { public:      d_String Name;
                           d_Set <d_Ref<PERS> > Angestellte   inverse PERS::Abteilung;

                           // Methoden:
                           void  Neuer_Angest (d_Ref<PERS> e) { Angestellte.insert_element (e); }
                           int   Arbeitet_Hier (d_Ref<PERS> e) { return Angestellte.contains_element (e);}
             };

class PROGRAMMIERER: public PERS
             {public:      d_Set <d_String> PSprachen;
                           // Konstruktor:
                           PROGRAMMIERER (d_String Name): PERS (Name) { };
            };
```

O_2: O_2C-Interface

- O_2-eigene Erweiterung von C
  - Erweiterung von C um benutzerdefinierbare Klassen, Mehrfachvererbung, Methoden etc.
  - Kollektionsklassen (*List*, *Set*, ...) mit Kontrollstrukturen: **for (**e **in** s **where** condition**) {**instructions**}**
  - Keine Unterscheidung zwischen Objekt und Referenz auf Objekt
  - Modularisierungskonzepte (Schemaimport und -export, programs, applications*)*

```
class PERS   type tuple ( public Name: string,
                          public birthday: date,
                          public Abteilung: ABT,
                          private Gehalt: integer)
             method
                          private init(name: string): PERS, // Aufruf z.B. durch: charly = new PERS("Charlie");
                          public getAge: integer,
end;

class ABT  type tuple ( public Name: string,
                        public Angestellte: set(PERS) )
             method
                        public Neuer_Angest (PERS e),  // -> Rumpf: Angestellte.insert_element (e);
                        public Arbeitet_Hier (PERS e): boolean, // -> Rumpf: return (e in Angestellte);
end;
```

O_2: Persistenz

- "Persistenz durch Erreichbarkeit"
- Ausgewählte Objekte werden explizit als persistente "Einstiegs"-Objekte definiert (*persistent roots*)
- Erreichbarkeitsregel: Jedes von einem Einstiegsobjekt aus erreichbare Objekt ist ebenfalls persistent
- Garbage collection: Löschen unerreichbarer Objekte (systemseitig oder explizit durch Benutzer)
- Definition von *persistent root objects* in O_2/C++:

```
d_Database myDatabase;
myDatabase.open("My Database");
// Definition von names (= persistent root objects)
myDatabase.create_persistent_root("myBestFriend", "d_Ref<PERS>");
myDatabase.create_persistent_root("persons", "d_Set<d_Ref<PERS> >");
...
d_Ref<Person> bestFriend("myBestFriend");
d_Set<d_Ref<PERS> > allPersons("persons");
```

- Persistent root objects in O_2C (definiert im Schema):

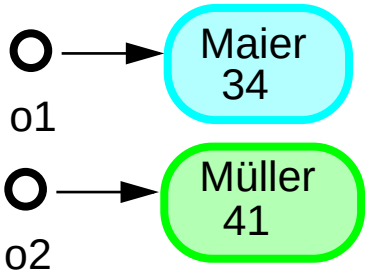
```
name persons set(PERS);
```

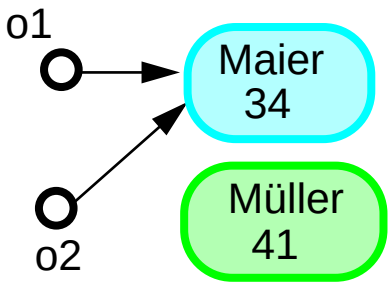
O_2: Programmfragment

```
#include “o2lib_CC.hxx”
#include <persdb.hxx>     /* Deklarationen PERS, ABT etc. */

d_Session  session; d_Database database;
main () {
        d_Transaction trans;

        session.set_default_env();
        if (session.begin(argc, argv, OL_GRAPHIC)) exit(1);
        database.open(“person_base”);
        database.create_persistent_root (“Ing_Abt”,”d_Ref<ABT>”);
        d_Ref<ABT> Ingenieur_Abt(“Ing_Abt”);
        trans.begin();
          d_Ref<PROGRAMMIERER> prog=new PROGRAMMIERER("Fred Baumann");
          Ingenieur_Abt.Neuer_Angest (prog); prog-> Alter = 30;
        trans.commit ();
        database.close();
        session.end();
        return 0;
}
```

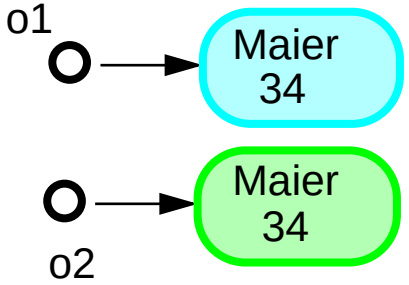
O_2

- Zuweisungen: Referenzsemantik vs. Wertesemantik



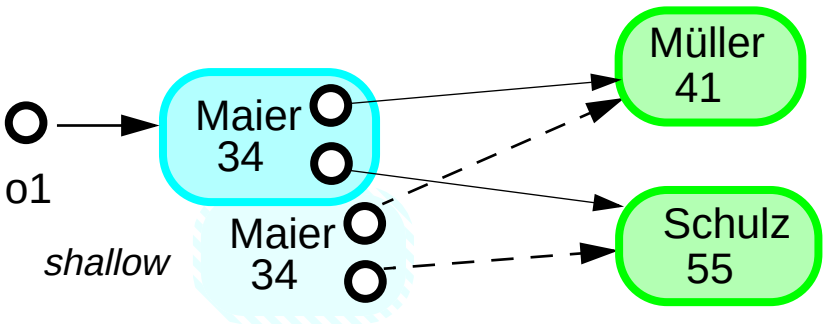


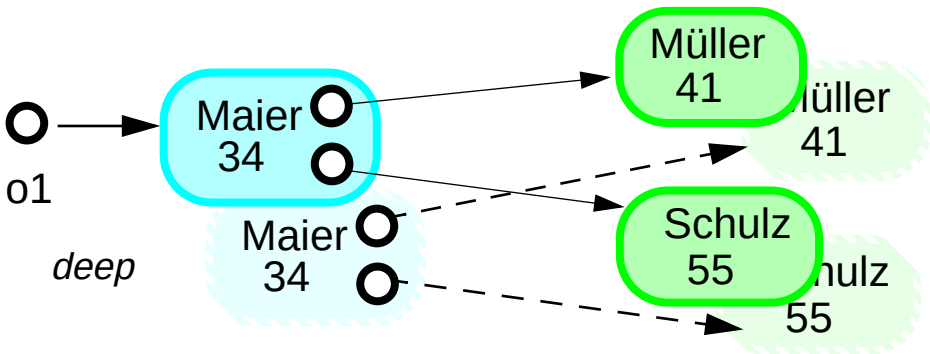
Referenzsemantik
(ändert Referenz von o2 auf o1)



Wertesemantik
(kopiert gesamten Zustand)

- O_2 unterstützt *flaches* und *tiefes Kopieren* (*copy* und *deep_copy*)





- Gleichheitsoperator (==) realisiert Test auf Identität, *equal* und *deep_equal* testen auf Gleichheit

O_2

- Operationen zur Datenbankverwaltung

  d_Database::create(...) **d_Database::destroy(...)**

  d_Database::open(...) **d_Database::close(...)**

  d_Database::create_persistent_root(...) **d_Database::garbage(...)**

- Zugriffe auf persistente Objekte müssen innerhalb von (ACID-) Transaktionen erfolgen

- O_2C-Programme können als *transaction* gekennzeichnet werden; andernfalls sind nur Leseoperationen im Programmrumpf erlaubt (ansonsten Laufzeitfehler!)

- in O_2/C++: explizite Beginn- und Ende-Operationen

  d_Transaction::begin () **d_Transaction::commit ()**

  d_Transaction::validate() **d_Transaction::abort ()**

- Unterstützung von geschachtelten Transaktionen
  - Transaktion kann aus Hierarchie von Sub-Transaktionen bestehen
  - umgebende Transaktion übernimmt Kontrolle über innere (Sub-) Transaktionen

O_2: Queries

- 4 Möglichkeiten für Abfragen auf Objekte
  - OQL-Aufrufe von O_2C bzw. O_2/C++-Programmen aus
  - deskriptiv über Selektionsausdrücke auf Kollektionen
  - navigierend über definierte DB-Einstiegspunkte (*named persistent root objects*)
  - interaktives OQL mit OQL-Shell

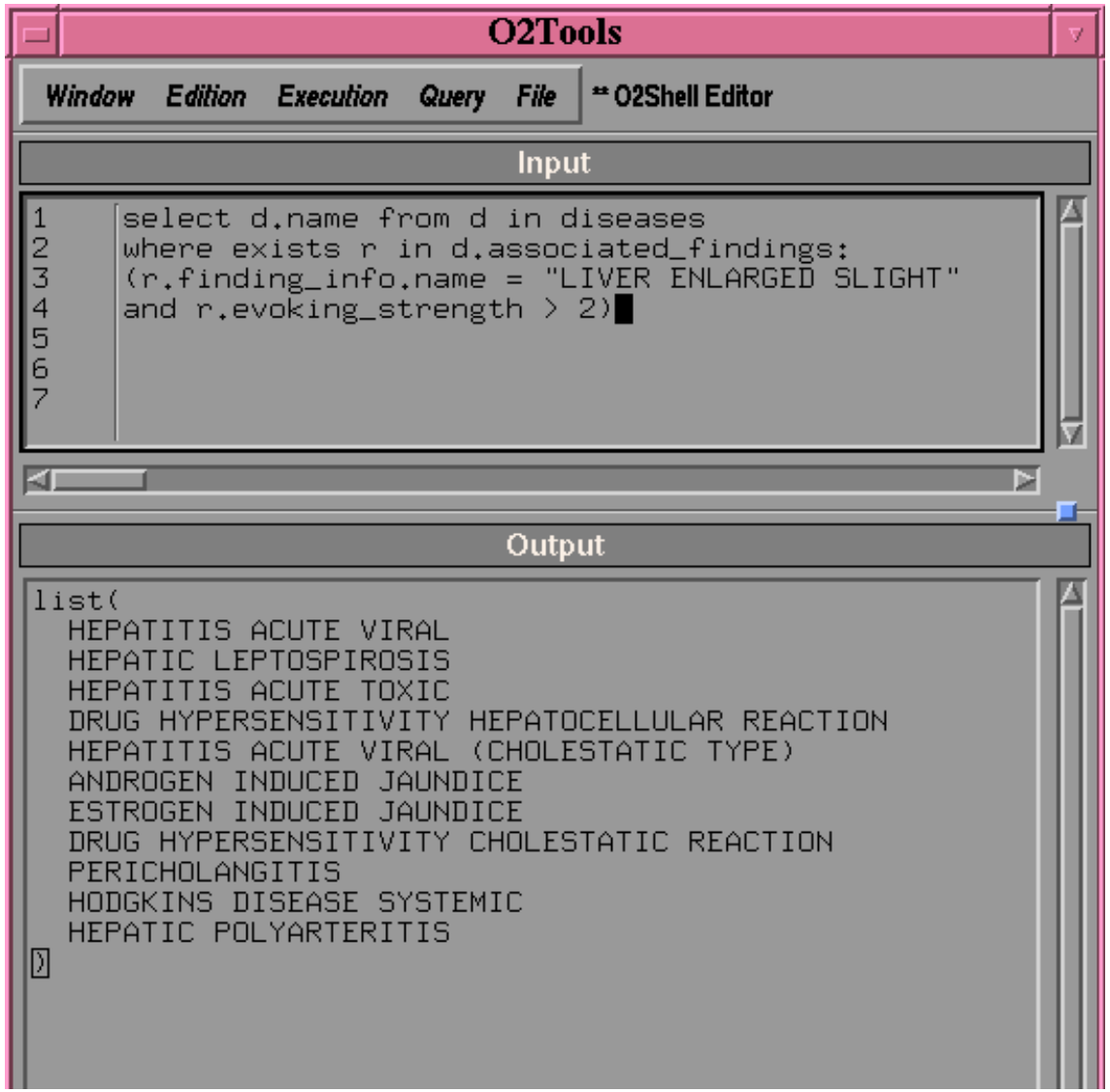
- OQL-Aufrufe von O_2C bzw. O_2/C++-Programmen aus:

```
d_Set<d_Ref<Person> > persons;
d_Set<d_Ref<Person> > result;
double salary = ...;
... // Auffüllen von persons
d_OQL_Query q1 (“select p from p in $1 where p.salary > $2”); // Query-Definition
q1<<persons<<salary; // Query-Parametrisierung
d_oql_execute (q1, result); // Query-Durchführung
```

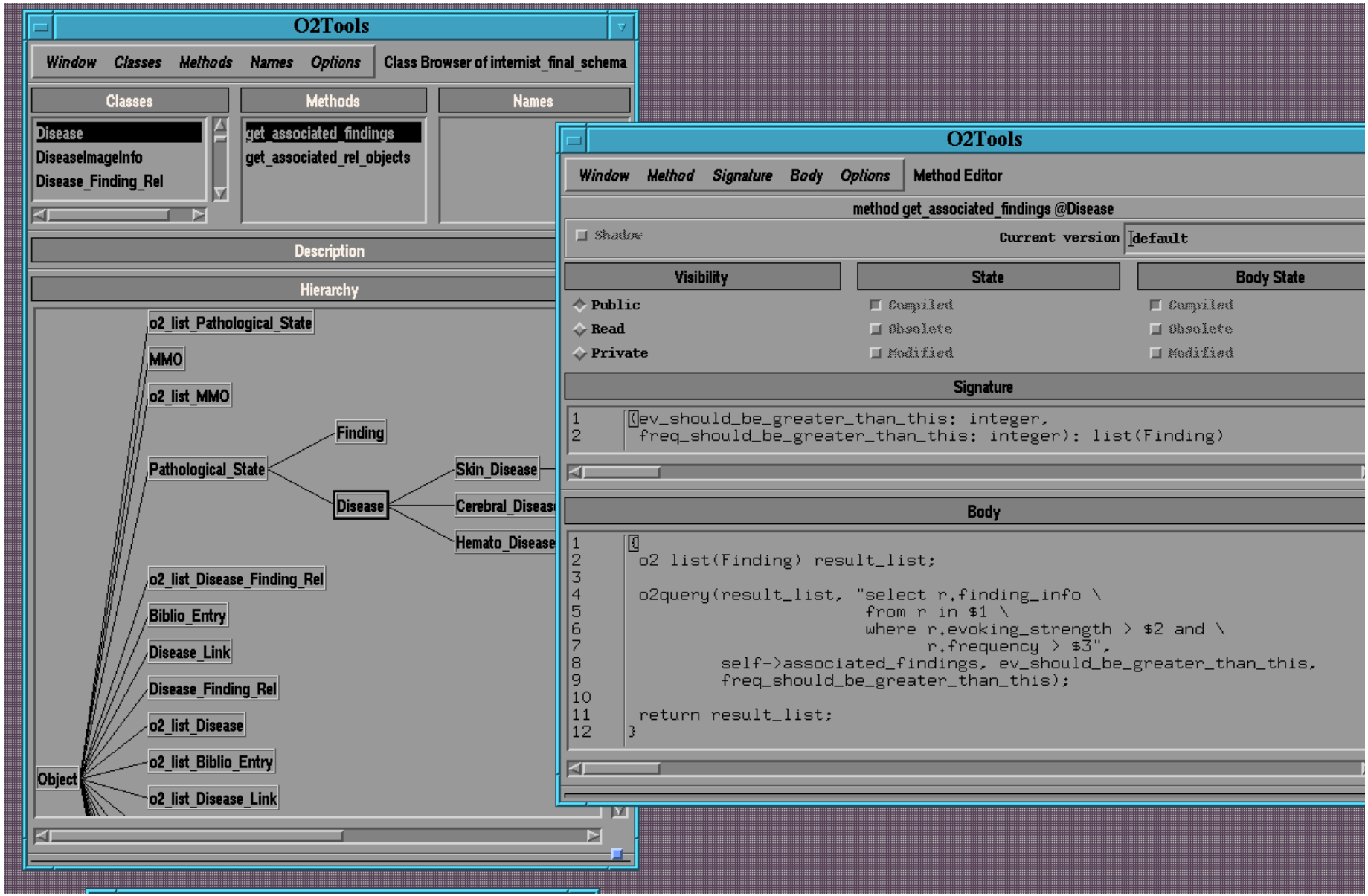
- Selektionsausdrücke auf Kollektionen

```
d_Set<d_Ref<Person> > boys;
people.query (boys, “this.age<13 and this.male=true”); // “this” ist Laufvariable auf Kollektion
```

O_2: Interaktives OQL



O_2: Graphische Programmierumgebung



Weitere Produkte (nicht vollständig)

- ObjectStore
- Poet
- Itasca
- Objectivity/DB
- Ontos
- Versant

5. Architektur von OODBS

- Workstation/Server-Architekturen
  - Page-Server
  - Objekt-Server
  - Query-Server

- Objekt-Zugriff (Pointer Swizzling)

- Unterstützung langer Entwurfsvorgänge
  - Checkout/Checkin
  - Unterstützung von Gruppenarbeit

Leistungsanforderungen für OODBS-Anwendungen

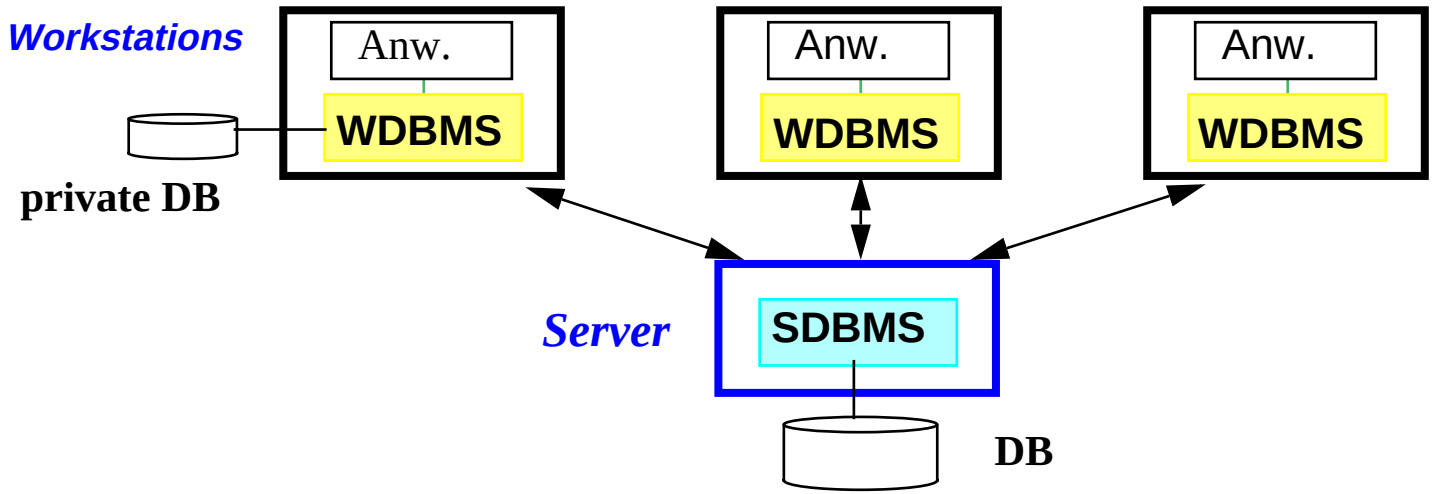
- schnelle Antwortzeiten
- komplexe Anwendungslogik
- Daten: großer Umfang, komplexe Struktur
- schnelle Navigation (Dereferenzierung)
- lange Dauer von Verarbeitungsvorgängen
- häufig wiederholter Zugriff auf Teilobjekte

⇒ **Workstation/Server-Architektur**

- weitgehend lokale Verarbeitung in WS
- Minimierung von Server-Zugriffen

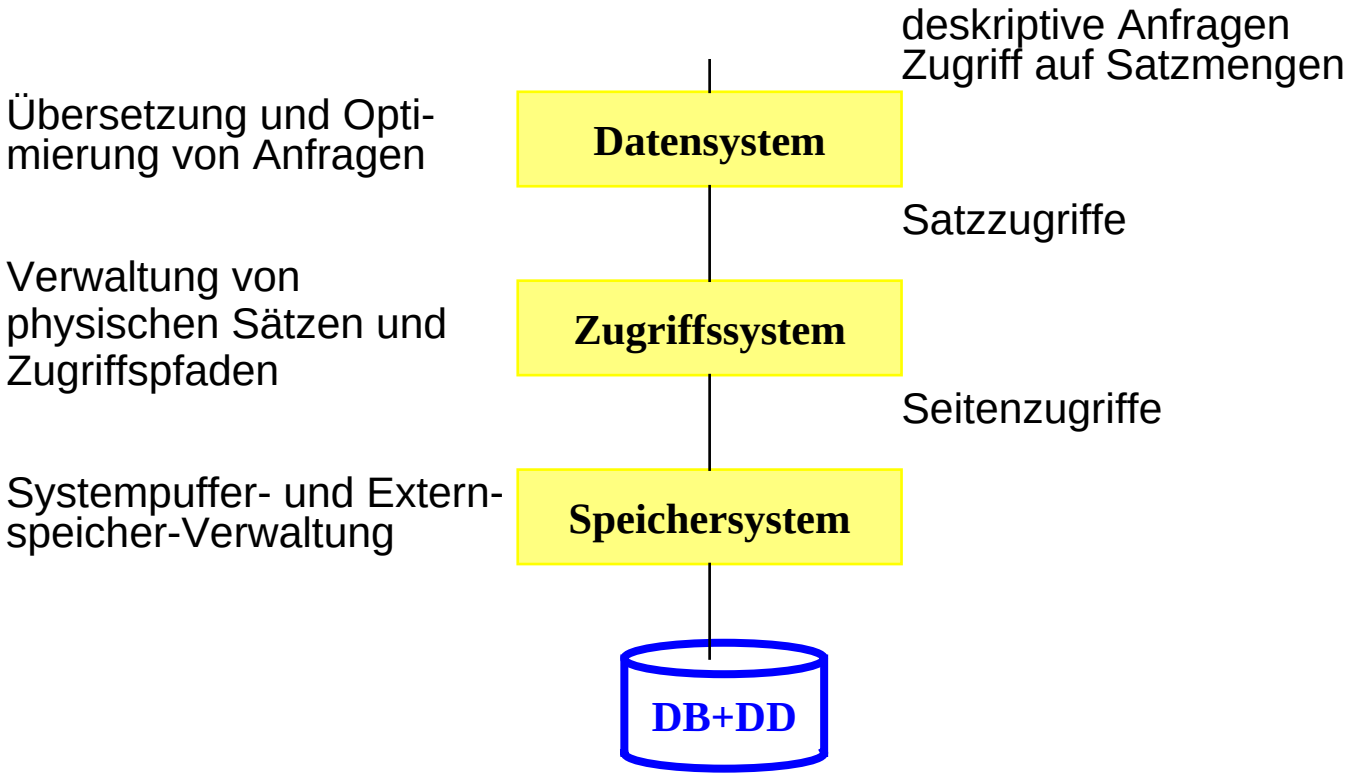
Workstation/Server-Architektur

- Funktionale Aufteilung in Server- und Workstation-DBMS
- Nutzung von Grafikeigenschaften und Verarbeitungskapazität leistungsfähiger Workstations



- Server bietet globale Dienste
  - Externspeicherverwaltung
  - Logging
  - Synchronisation, etc.
- Verarbeitung von DB-Operationen durch Workstation-DBMS
  - benötigte Objekte werden im Hauptspeicher der Workstation gepuffert (Objektpuffer)
  - Nutzung von Lokalität zur Einsparung von Kommunikationsvorgängen

Aufbau eines zentralisierten DBS

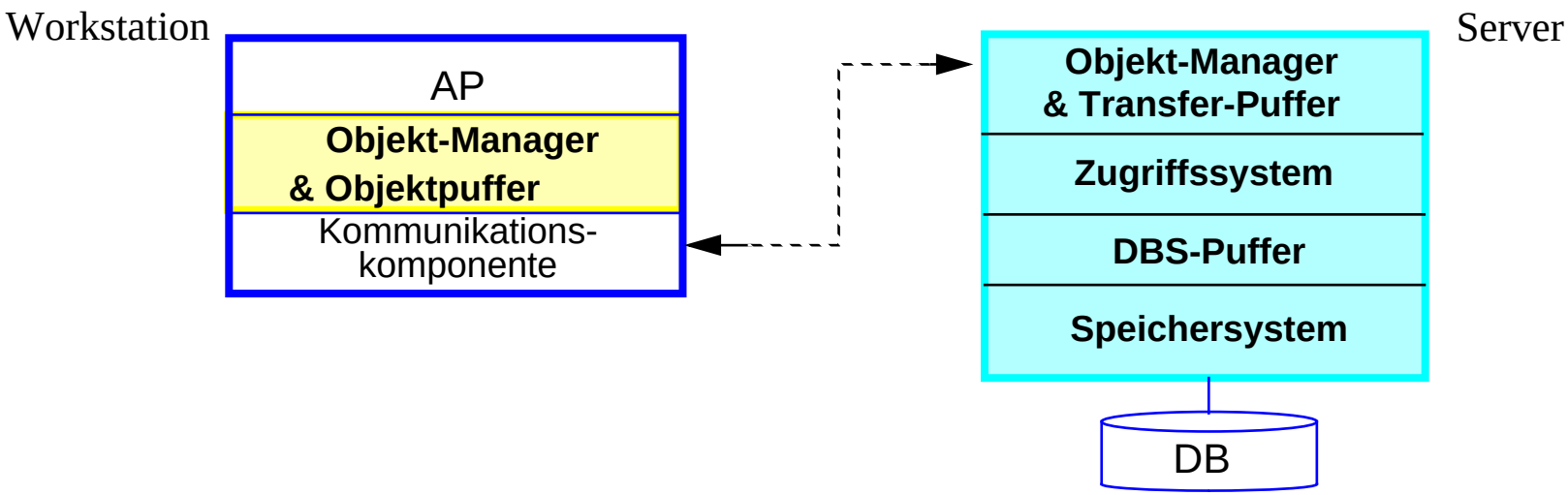


- zusätzliche Funktionen zur Transaktionsverwaltung (Synchronisation, Logging/Recovery, Integritätssicherung)
- Wie soll die DBMS-Funktionalität, insbesondere Speichersystem, Zugriffssystem, Datensystem/Objekt-Manager, auf eine verteilte Umgebung (n Clients und Server) abgebildet werden?

Überblick von WS/Server-Architekturen

- Query-Server
  - gesamtes Ergebnis einer Anfrage wird auf einmal zurückgeliefert (mengenorientierte Anforderung)
  - Unterscheidung nach
    - Mengen einfacher Objekte (SQL)
    - Mengen komplexer Objekte
- Objekt-Server
  - es wird ein Objekt nach dem anderen an den Client geliefert (one object at a time)
  - einfache oder komplexe Objekte
- Page-Server
  - es werden Seiten nacheinander angefordert und einzeln übertragen
- File-Server
  - Das Network File System (NFS) dient als Speichersystem
  - zusätzlich wird eine Transaktionsverwaltung benötigt
- Verteilte OODBS
  - volle DBMS-Funktionalität pro Rechner (Workstation)
  - gleichberechtigte Anfrageverarbeitung
  - Datenverteilung unter allen Rechnern

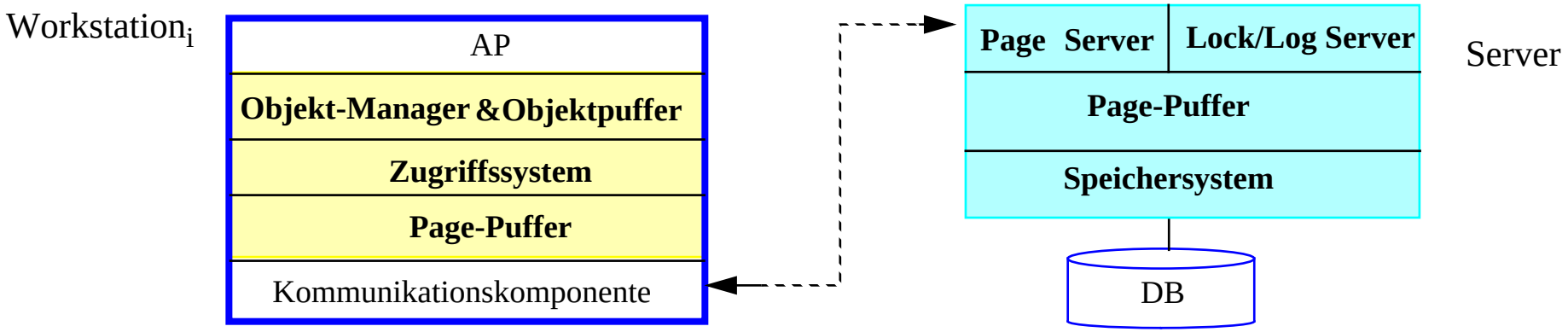
Query-Server und Objekt-Server



- zweistufige Verarbeitung
  - gleiche Funktionen in WS und Server
  - Replikation der Daten in WS und Server
  - Übertragung nur der benötigten Objekte
  - Duplikation aller Änderungsoperationen

- weitere Eigenschaften
  - Objekt-Server: hoher Kommunikationsaufwand durch objektweises Holen der referenzierten Objekte
  - Objekte können im DB-Puffer, im Objektpuffer oder auf Platte sein: Erstellen eines kohärenten Verarbeitungskontexts
  - Zurückschreiben in DB-Puffer oder Aktualisierung des Objektpuffers kann für Methodenausführung erforderlich sein

Page-Server



- einstufige Verarbeitung
  - höhere Funktionen (obere Schichten) laufen nur in der WS
  - in der Regel navigierende Schnittstelle
  - Transfereinheit: Seite → Seitensperren
  - Leistungsfähigkeit wird erheblich durch Clusterbildung bestimmt
- Pufferung in der WS: Seiten und/oder Objekte
- weitere Eigenschaften
  - gesamte Objektverarbeitung in der WS
  - Server: Concurrency Control + Recovery
  - Transfer von 4 KB nur wenig teurer als der von 100 B (mehrere Objekte)
  - mehr WS pro Server möglich
  - Methoden können nur in WS ausgeführt werden (sequentieller Scan!)

Vergleich

	Query-Server	Objekt-Server	Page-Server
Methodenausführung			
Integritätsbedingungen			
Kommunikationsaufwand			
Skalierbarkeit			
Realisierungsaufwand			

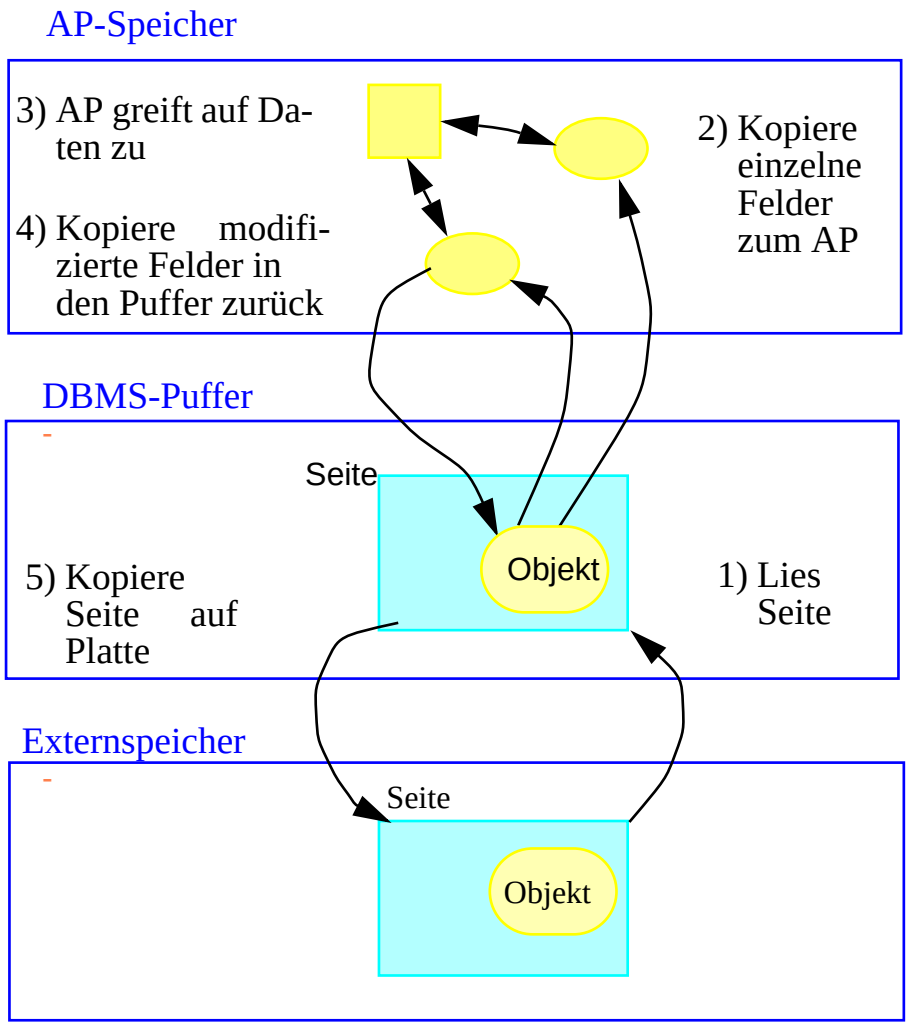
- Literaturhinweis: T. Härder et al.:
  Workstation-Server-Architekturen für datenbankbasierte Ingenieuranwendungen.
  Informatik - Forschung und Entwicklung 1995 (2), S. 55-72

Objekt-Pufferung und -Zugriff

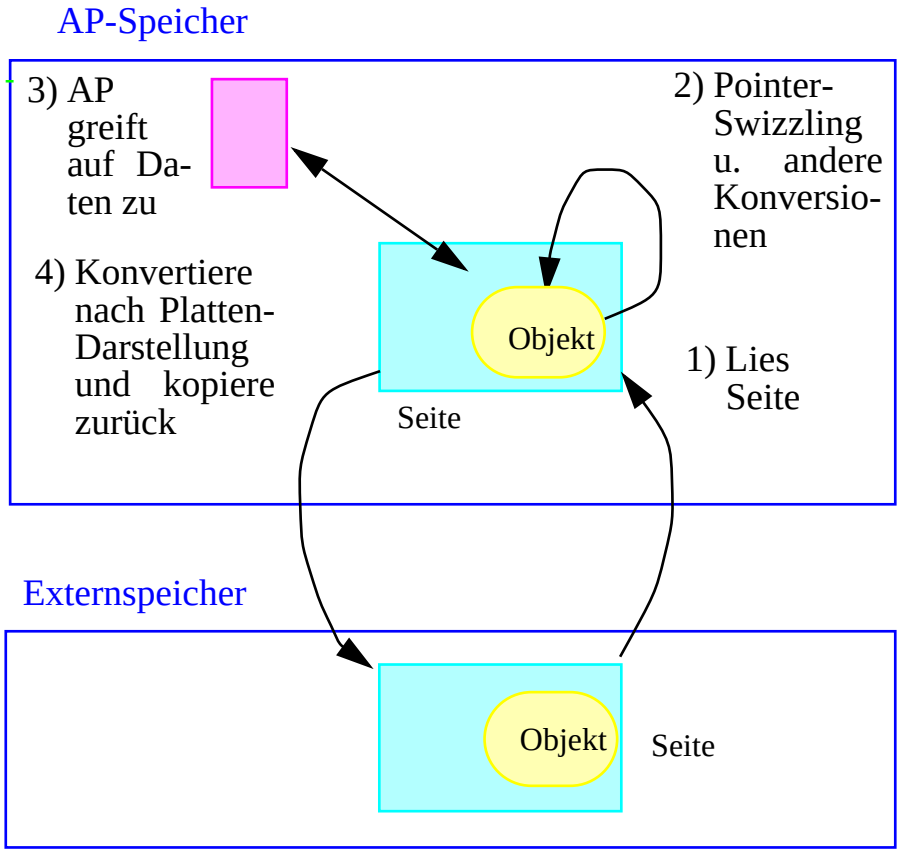
- Leistungsziel: 10^5 Objektreferenzen pro sec bei CAD-Anwendungen
- Referenzlokalität und “direkter” Objektzugriff sind die kritischen, leistungsbestimmenden Faktoren
- Kosten für einfachste Datenoperationen:
  - herkömmliche DBMS-Realisierung > ms
  - gute Integration durch DBPL < µs
- hohe Leistungsfähigkeit erfordert
  - Hauptspeicherpufferung des Working Sets (Objektpuffer)
  - Pointer/OID-Swizzling: Referenzumsetzung oder schnelle OID-Abbildung
  - Kostenverlagerung von der Laufzeit zur Compile-Zeit
  - schnelle Abbildung von der Repräsentation auf Externspeicher zur Sprachrepräsentation der Anwendung (z. B. C++)
- bei Workstation/Server-Architekturen ist der Transport übers Netz besonders kritisch
  - mengenorientierte Server-Zugriffe
  - Minimierung der Datenübertragungsvorgänge (Granulate sind Serveranforderungen und Anfrageergebnis)

Objekt-Zugriff

traditioneller Ansatz



integrierter Ansatz (DBPL-Zugriff)



DBS2

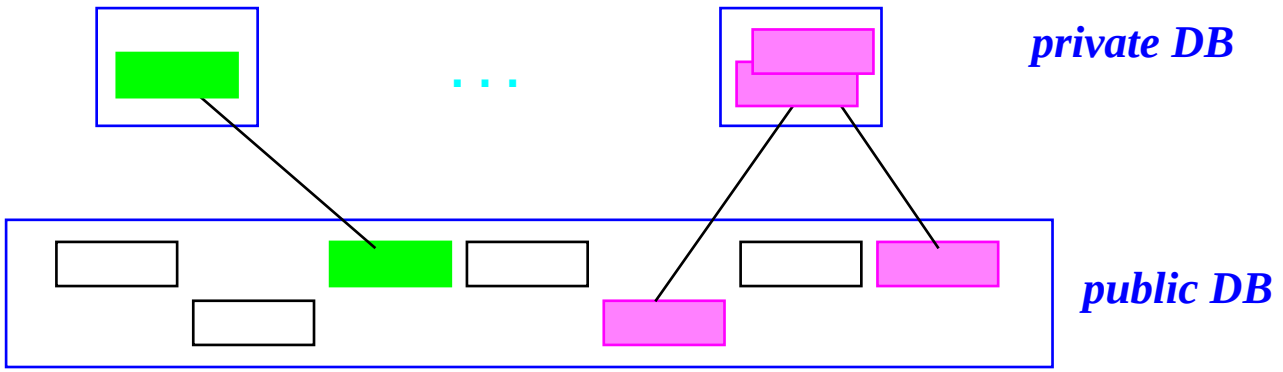
Transaktionsverwaltung

- Traditionelles Transaktionskonzept (ACID)
  - Atomicity
  - Consistency
  - Isolation
  - Durability

- Beschränkungen
  - auf kurze Transaktionen zugeschnitten, Probleme mit "lang-lebigen" Aktivitäten
  - Alles-oder-Nichts-Eigenschaft oft inakzeptabel: hoher Arbeitsverlust
  - Isolation
    - Leistungsprobleme durch "lange" Sperren
    - fehlende Unterstützung zur Kooperation
  - keine Binnenstruktur ("flache" Transaktionen)
  - keine Unterstützung zur Parallelisierung
  - fehlende Benutzerkontrolle

DB-Verarbeitung in Entwurfsumgebungen

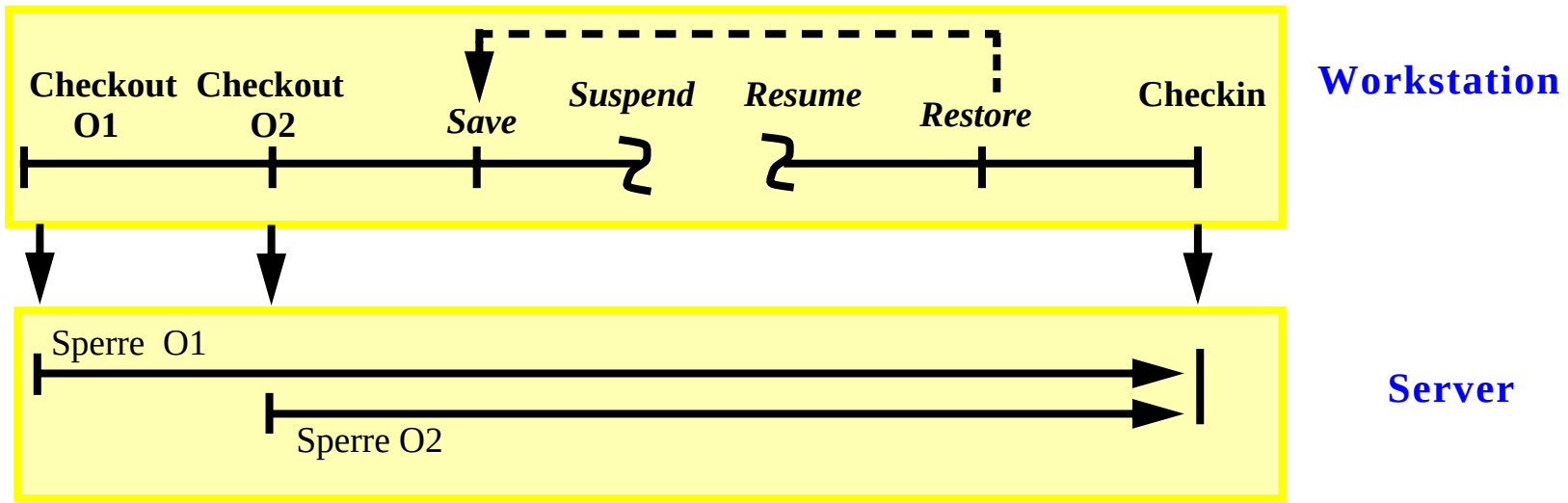
- Merkmale
  - Lange Dauer von Entwurfsvorgängen (Wochen/Monate)
  - Kontrollierte Kooperation zwischen mehreren Entwerfern
  - Unterstützung von Versionen
  - Benutzerkontrolle (nicht-deterministischer Ablauf)
- Checkout/Checkin-Modell
  - 1 public DB, n private DB (Änderungen in privater DB)
  - CHECKOUT: Kopieren des Entwurfsobjektes (public DB → private DB)
  - CHECKIN: (atomares) Einbringen der Änderungen am Ende der Entwurfstransaktion (private → public DB)



- nach CHECKOUT bleiben Objekte in der public DB gesperrt (permanente Langzeitsperren)
  - keine parallelen Änderungen
  - Lesen der ungeänderten Objekte weiterhin möglich
  - Verarbeitung im Einbenutzerbetrieb

Erweiterung des Modells

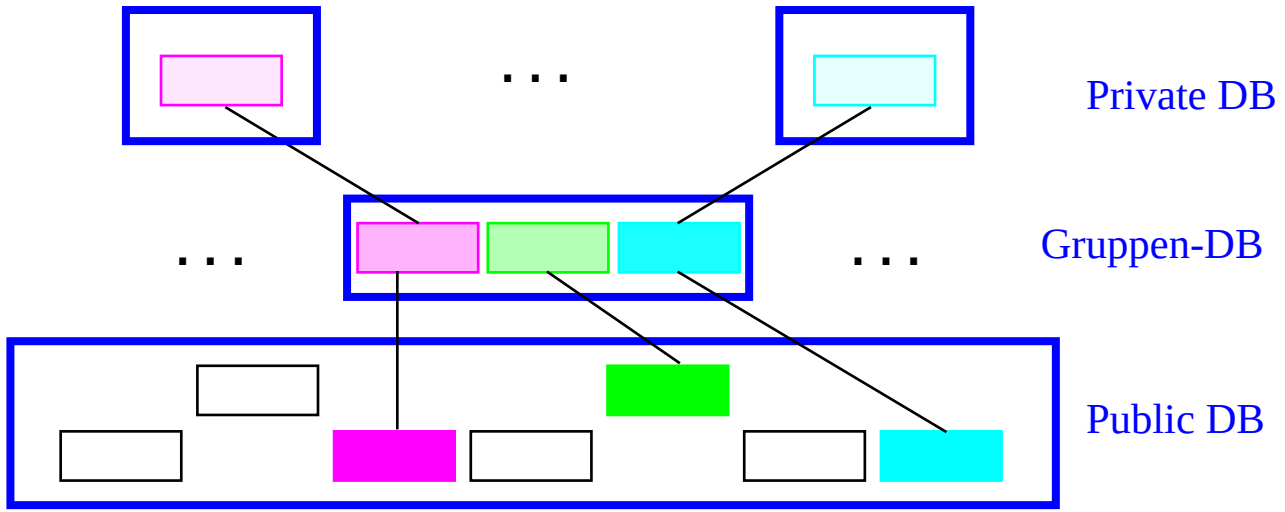
- Einführung von transaktionsinternen Rücksetzpunkten
  - benutzerdefinierte Savepoints (Operationen SAVE/RESTORE)
  - systemgenerierte Rücksetzpunkte (recovery points)
- SUSPEND/RESUME-Operationen zur Unterbrechung/Fortführung der Verarbeitung



- Reduzierung der Isolation:
  - Versionenkonzept
  - anwendungsbezogene Synchronisation
  - benutzergesteuerte Konfliktbehandlung

Unterstützung von Gruppenarbeit

- Gruppen- und Benutzertransaktionen
  - kooperative Transaktionshierarchien
  - keine “vollständigen” ACID-Eigenschaften
- Austausch vorläufiger Entwurfsdaten über Gruppen-DB

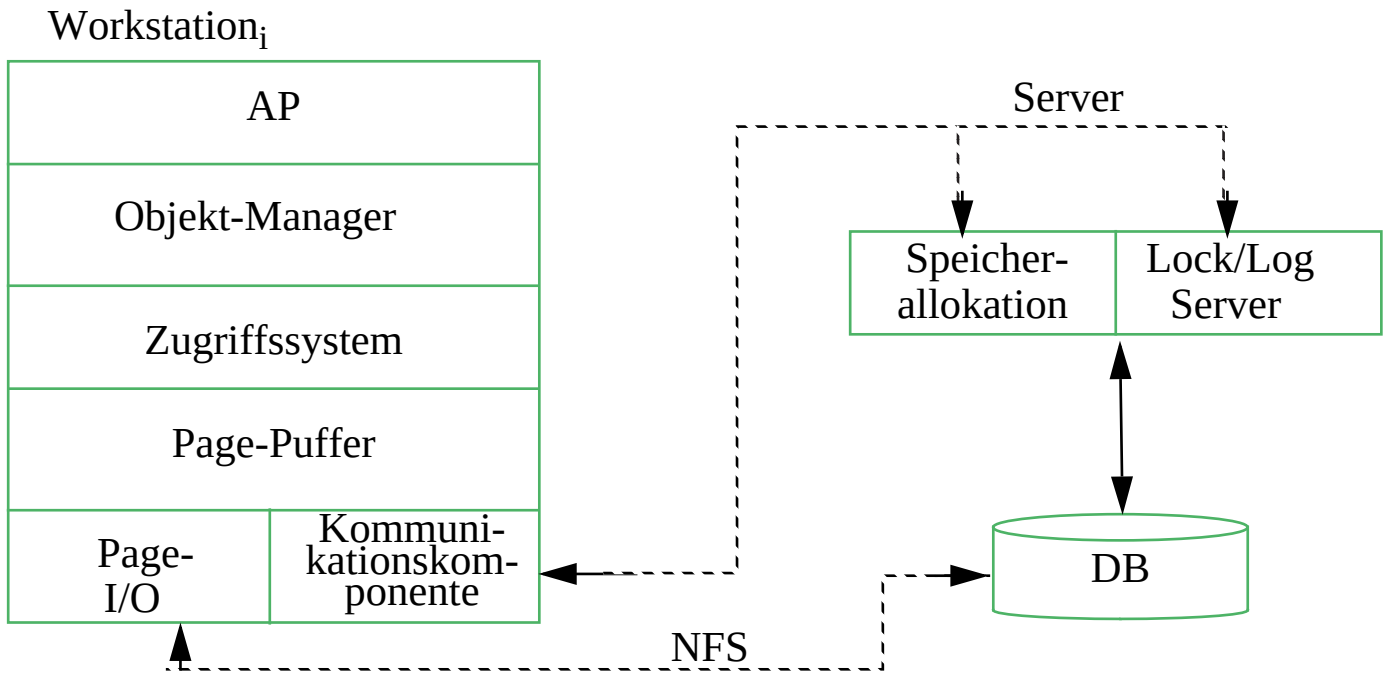


- 2-stufiges Checkout/Checkin-Konzept
  - Checkout/Checkin zwischen Public DB und Gruppen-DB sowie zwischen Gruppen-DB und Privaten DB
  - Benutzertransaktionen greifen auf Gruppen-DB zu
  - spezielle Operationen zum Datenaustausch innerhalb einer Gruppe

Zusammenfassung

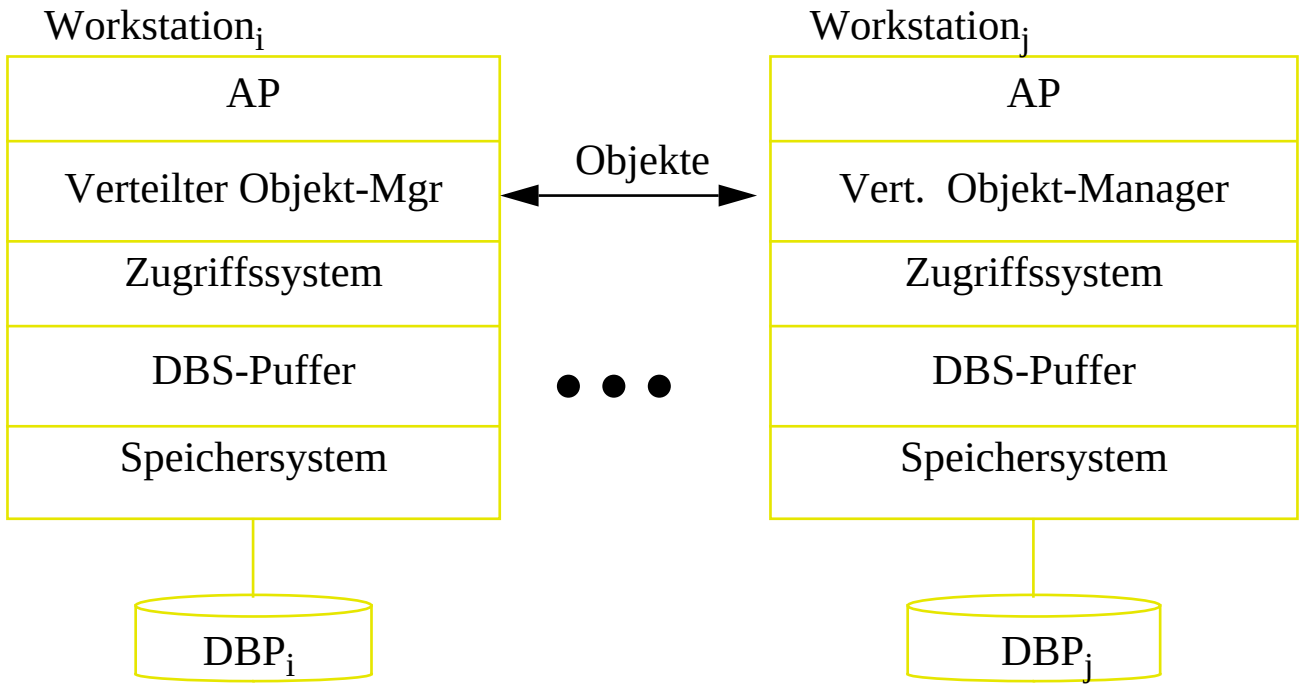
- Workstation/Server-Architekturen
  - Nutzung leistungsfähiger Workstations
  - Workstation-Pufferung von komplexen Objekten
  - Minimierung von Server-Zugriffen
- Hauptalternativen: Query- und Page-Server
- Pointer-Swizzling
  - Umsetzung von OIDs in Hauptspeicheradressen
  - sehr schnelle Navigation
- Lange Entwurfsvorgänge
  - ACID inakzeptabel
  - Checkout/Checkin-Modell
  - Unterstützung von Gruppenarbeit, Versionen etc.

File-Server



- Benutzung eines File-Servers (z. B. NFS), um DB-Seiten direkt zu lesen und zu schreiben
- Minimierung des Overheads auf dem Server
- Nachteile (wie Page-Server)
  - Server und NFS sind verschiedene Komponenten an verschiedenen Orten
  - NFS-E/A ist langsam (besonders Schreiben)
  - Concurrency Control und Recovery müssen auf Server abgewickelt werden
  - Anforderung von Gruppen von Seiten kann manche Nachteile vermindern

Verteiltes OODBS



- Jeder Knoten (WS) besitzt volle Funktionalität
  - Â hohe Systemkomplexität pro Knoten

- Verteilte Verarbeitung einer Transaktion
  - Weiterleiten von Anforderungen (Teiloperationen)
  - mengenorientierte Übertragung des Ergebnisses
  - verteiltes Commit-Protokoll
  - Problem der Datenpartitionierung (nimmt mit der Anzahl der Workstations zu)


DBS2

6. Objektrelationale DBS / SQL3

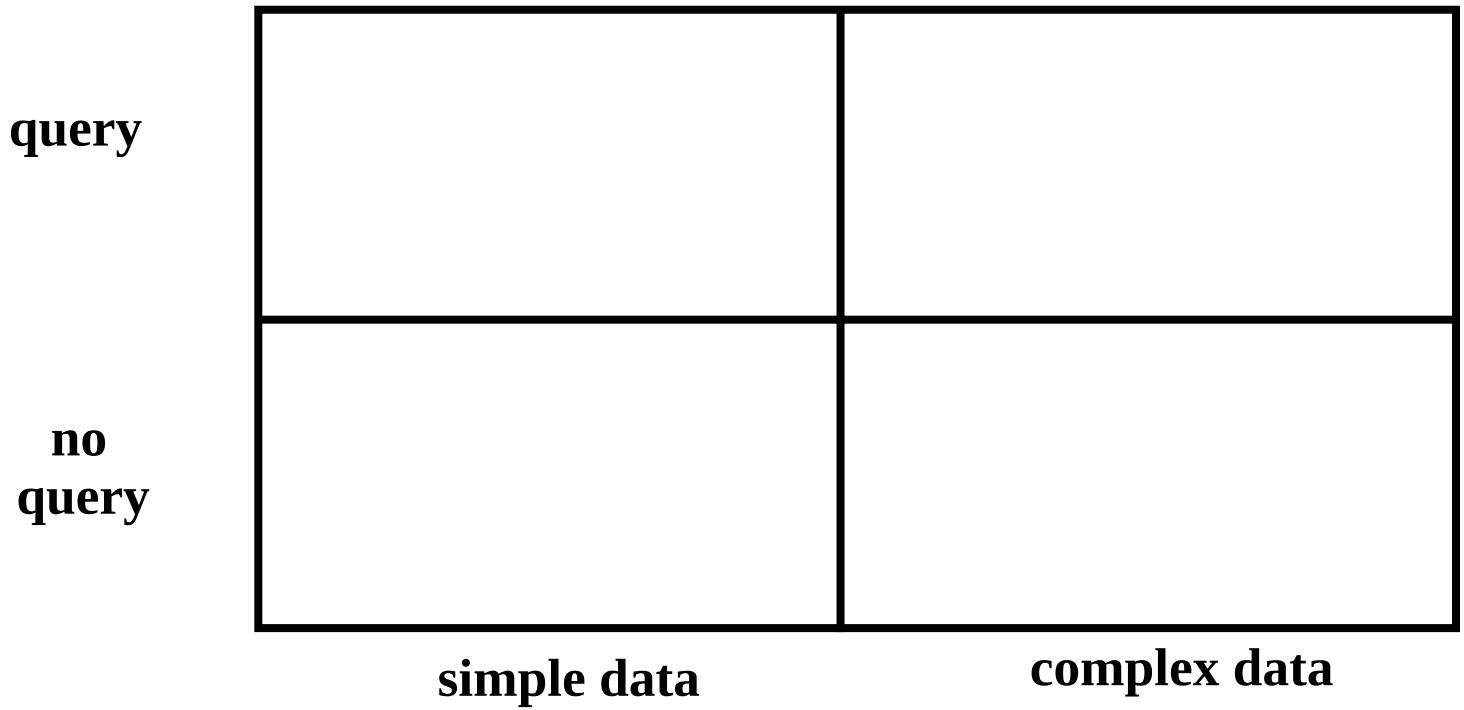
- Objektrelationale DBS vs. objektorientierte vs. relationale DBS
- SQL3-Standardisierung
- SQL3-Typsystem
  - Large Objects
  - benutzedefinierte Typen und Funktionen (UDTs, UDFs)
  - REF-Typen
  - DISTINCT-Typen
  - ROW-Typen
  - Typhierarchien / Tabellenhierarchien
- Rekursive Anfragen in SQL3
- ORDBS-Produkte (“Universal Server”)
  - Informix
  - Oracle

Objektrelationale DBS (ORDBS)

- Merkmale von ORDBS
  - Erweiterung des relationalen Datenmodells um Objekt-Orientierung (SQL3)
  - Erweiterbarkeit über benutzerdefinierte Datentypen (u.a. Multimedia-Datentypen) und benutzerdefinierte Funktionen
  - komplexe, nicht-atomare Attributtypen (z.B. relationenwertige Attribute)
  - Bewahrung der Grundlagen relationaler DBS, insbesonderer deklarativer Datenzugriff, Sichtkonzept etc.

- Vergleich
  - relationale DBS: einfache Datentypen, Queries, ...
  - OODBS basierend auf persistenten Programmiersprachen: komplexe Datentypen, gute PS-Integration, hohe Leistung für navigierende Zugriffe
  - ORDBS: komplexe Datentypen, Querying ...

Grobvergleich nach Stonebraker*



- kein Systemansatz erfüllt alle Anforderungen gleichermaßen gut
- OODBS im Stile von persistenten Programmiersprachen bleiben Nischen-Produkte
- Bedeutung von ORDBS nimmt stark zu

* M. Stonebraker, P. Brown: Object-Relational DBMS - The Next Great Wave. Morgan Kaufmann, 1996; 2nd ed. 1998

SQL-Standardisierung*

1986 SQL86

- DDL-Anweisungen für Relationen, Sichten, Zugriffsrechte
- DML-Anweisungen für Standard-Operationen
- keine Integritätszusicherungen

1989 SQL89 (120 Seiten)

- Revision von SQL86
- DEFAULT-Klausel (Wertzuweisung, verschieden von NULL)
- CHECK-Bedingung (rudimentäres Domain-Konzept)
- Basiskonzept der Referentiellen Integrität (Referenzen auf Primärschlüssel und Schlüsselkandidaten)

1992 SQL92 (SQL2)

- Entry Level (~ SQL89 + gerinfügige Erweiterungen und Korrekturen)
- Intermediate Level
  (Dynamic SQL, Join-Varianten, CASCADE DELETE, Domains ...)
- Full Level (580 Seiten)
  (CASCADE UPDATE, MATCH-Klausel, Subquery in CHECK, Assertions, DEFERRED ...)

1995/96Nachträge zu SQL-92: Call-Level-Interface / Persistent Stored Modules (Stored Procedures)

1999 SQL3 bzw. SQL:1999

> 2000 SQL4

* http://www.jcc.com/SQLPages/jccs_sql.htm

Aufbau des SQL3-Standards

Part 1: Framework (beschreibt Aufbau des Standards)

Part 2: Foundation (Kern)

- umfassendes und erweiterbares Typsystem (benutzerdef. Datentypen und Funktionen, zusätzliche Built-in-Datentypen etc.)
- Vererbung und Overloading
- Trigger, Rekursion (Berechnung der transitiven Hülle)
- asynchrone Ausführung von Operationen, Savepoints, Rollen, . . .

Part 3: SQL/CLI (Call Level Interface)

Part 4: SQL/PSM (Persistent Storage Modules)

- gespeicherte Prozeduren / Funktionen (stored procedures)
- operationale Vollständigkeit durch zusätzliche Kontrollanweisungen; Ausnahmebehandlung
- Performance-Verbesserung für zentralisierte DBS und Client/Server-Umgebungen

Part 5: SQL/Bindings

Part 6: SQL/Transaction

- Schnittstellenbeschreibung zur verteilten Transaktionsverarbeitung nach X/Open (XA-Schnittstelle)
- API-Festlegung für Transaction Manager

Part 7: SQL/Temporal

Part 9: SQL/MED (Management of External Data)

Part 10: SQL/OLB (Object Language Binding), entspricht SQLJ

SQL3-Standardisierung*

- nationale (ANSI, DIN ...) und internationale Standardisierung (ISO)
  - internationale Zuständigkeit bei ISO/IEC Joint Technical Committee JTC1, Subcommittee SC32 Working Group WG3 (SQL) bzw. WG4 (SQL Multimedia)
  - ANSI: NCITS (National Committee for Information Technology Standardization) Technical Committee H2 (früher: X3H2)
  - Standard-Bezeichnung: ISO/IEC 9075:1999, Information technology - Database languages - SQL: Part ...
- ISO-Standardisierungsschritte
  - interne Dokumente
  - Working Drafts
  - Committee Draft (CD) -> Public Review
  - Draft International Standard (DIS)
  - International Standard (IS)
- SQL92-Nachträge CLI und PSM wurden 1995/96 bereits als Standards verabschiedet
- Aktuelle Zeitplanung SQL3
  - derzeit DIS
  - IS für 1999 geplant (Part 1-5) inklusive überarbeiteter Version von CLI -95

* A. Eisenberg, J. Melton: SQL:1999, formerly known as SQL3, ACM SIGMOD Record, 1999 (1), March 1999

Typsystem von SQL3

- erweiterbares Typkonzept
  - vordefinierte Datentypen
  - konstruierte Typen (Konstruktoren): REF, Tupel-Typen (Row Type), Kollektionstypen (z.Zt. nur ARRAY)
  - benutzerdefinierte Datentypen (user-defined types, UDT): Structured Types und Distinct Types
- Definition von UDTs unter Verwendung von vordefinierten Typen, konstruierten Typen und vorher definierten UDTs
- UDTs unterstützen
  - Kapselung
  - Vererbung (Subtypen)
  - Overloading
- alle Daten werden weiterhin innerhalb von Tabellen gehalten
  - Definition von Tabellen auf Basis von strukturierten UDTs und Tupel-Typen möglich
  - Bildung von Subtabellen (analog zu UDT-Subtypen)
- neue vordefinierte Datentypen
  - Boolean
  - Large Objects (Binärdarstellung bzw. Texte)

Large Objects

- 2 neue Datentypen: BLOB (Binary Large Object) und CLOB (Character Large Object)
- Verwaltung großer Objekte im DBS (nicht in separaten Dateien)
  - umgeht große Datentransfers und Pufferung durch Anwendung
  - Zugriff auf Teilbereiche

```
CREATE TABLE Pers    (PNR          INTEGER,
                      Name         VARCHAR (40),
                      Lebenslauf   CLOB (75K),
                      Unterschrift BLOB (1M),
                      Bild         BLOB (12M))
```

- einige Operationen sind auf LOBs nicht möglich :
  Schlüsselbedingung, Kleiner/Größer-Vergleiche, Sortierung (ORDER BY, GROUP BY)
- unterstützte Operationen
  - Suchen und Ersetzen von Werten (bzw. partiellen Werten)
  - LIKE-Prädikate
  - Konkatenation, SUBSTRING, LENGTH, ...
- indirekte Verarbeitung großer Objekte über Locator-Konzept (ohne Datentransfer zur Anwendung)

Syntax der UDT-Definition (vereinfacht)

CREATE TYPE <UDT name> [[<subtype clause>] [AS <representation>] [<instantiable clause>]
<finality> [<reference type specification>] [<cast option>] [<method specification list>]

<subtype clause> ::= UNDER <supertype name>

<representation> ::= <predefined type> | [**(** <member> , ... **)**]

<instantiable clause> ::= INSTANTIABLE | NOT INSTANTIABLE

<finality> ::= FINAL | NOT FINAL

<member> ::= <attribute definition>

<reference type specification> ::= REF USING <predefined type> | REF <list of attributes> |
REF IS SYSTEM GENERATED

<method specification> ::= <original method specification> | <overriding method specification>

<original method specification> ::= <partial method specification> <routine characteristics>

<overriding method specification> ::= OVERRIDING <partial method specification>

<partial method specification> ::= [INSTANCE | STATIC] METHOD <routine name>
<SQL parameter declaration list> <returns clause>

- Definition von DISTINCT-Typen (<representation> entspricht <predefined type>) und strukturierten Typen
- Aufbau von Typhierarchien (Subtyp-Klausel): nur einfache Vererbung
- Spezifikation von Datenelementen (Attributen) und Methoden

UDT-Beispiel

```
CREATE TYPE Adresse
        (Strasse        VARCHAR (40),
         PLZ            CHAR (5),
         Ort            VARCHAR (40) );

CREATE TYPE PersonT
        (PNR            int,
         Name           VARCHAR (40),
        Manager         REF (PersonT),
        Anschrift       Adresse,
        Grundgehalt     REAL,
        Provision       REAL )
        INSTANTIABLE
        NOT FINAL
        REF (PNR)
        INSTANCE METHOD Einkommen () RETURNS REAL);
```

Funktionen und Methoden

- Routinen (Funktionen und Prozeduren) als eigenständige Schemaobjekte bereits in SQL/PSM

	SQL-Routinen	Externe Routinen
Aufruf in SQL (SQL-invoked routines)	SQL-Funktionen (inkl. Methoden) und SQL-Prozeduren	externe Funktionen und Prozeduren
Externer Aufruf	nur SQL-Prozeduren (keine Funktionen)	(nicht relevant für SQL)

- Methoden: beziehen sich auf genau einen UDT
- Realisierung aller Routinen und Methoden über prozedurale SQL-Spracherweiterungen oder in externer Programmiersprache (C, Java, ...)
- Namen von SQL-Routinen und Methoden können überladen werden (keine Eindeutigkeit erforderlich)
  - bei SQL-Routinen wird zur Übersetzungszeit anhand der Anzahl und Typen der Parameter bereits die am “besten passende” Routine ausgewählt
  - bei Methoden wird dynamisches Binden zur Laufzeit unterstützt

UDT-Kapselung

- vollständige Kapselung
- Attributzugriff erfolgt ausschließlich über Funktionen (Methoden)
  - keine Unterscheidung zwischen Attributzugriff und Funktionsaufruf
  - sichtbare UDT-Schnittstelle besteht aus Menge von Funktionen/Methoden
- für jedes Attribut wird implizit eine Funktion zum Lesen (Observer) sowie zum Ändern (Mutator) erzeugt
- implzit erzeugte Funktionen für UDT Adresse

Observer-Funktionen:

```
FUNCTION   Strasse (Adresse)          RETURNS VARCHAR (40);
FUNCTION   PLZ (Adresse)              RETURNS CHAR (5);
FUNCTION   Ort (Adresse)              RETURNS VARCHAR (40);
```

Mutator-Funktionen:

```
FUNCTION   Strasse (Adresse, VARCHAR (40))   RETURNS Adresse;
FUNCTION   PLZ (Adresse, CHAR(5))            RETURNS Adresse;
FUNCTION   Ort (Adresse, VARCHAR (40) )      RETURNS Adresse;
```

Punkt (dot)-Notation

- für Attributzugriff kann wahlweise Funktionsaufrufe oder Punkt-Notation (.) verwendet werden
  - Punkt-Notation entspricht “syntaktischem Zucker” für Funktionsaufrufe

a.x	ist äquivalent zu	x (a)
SET a.x = y	ist äquivalent zu	x (a, y)

- Punkt-Notation kann mehrstufig angewendet werden (a.b.c.d.e)

```
BEGIN
        DECLARE p       PersonT;
         ...
        SET p.Grundgehalt = 2500.66;
        SET x = p.Grundgehalt;

        SET y = p.Anschrift..Ort;
        SET p.Anschrift.PLZ = “04109”;
        ...
```

- erlaubt navigierende Zugriffe (Pfadausdrücke)
- **Aber: für Methodenaufrufe ist Punkt-Notation obligatorisch!**

Initialisierung von UDT-Instanzen

- DBS stellt für jeden instantiierbaren UDT Default-Konstruktor zum Erzeugen von UDT-Instanzen bereit

  CREATE FUNCTION PersonT () RETURNS PersonT

- Initialisierung kann durch Mutator-Funktionen erfolgen

- Benutzer kann beliebige Anzahl eigener Konstruktoren definieren, um Initialisierung beim Anlegen des Objektes zu erreichen (über Parameter)

```
CREATE FUNCTION PersonT (n varchar(40), a Adresse) RETURNS PersonT
BEGIN
        DECLARE p      PersonT;
        SET p = PersonT();
        SET p.Name = n;
        SET p.Anschrift = a;
        RETURN p;
END;
```

UDT-Einsatz

- UDTs können überall verwendet werden, wo vordefinierte Datentypen in SQL zum Einsatz kommen
  - als Attributtyp anderer UDTs
  - als Parametertyp von Funktionen und Prozeduren
  - als Typ von SQL-Variablen
  - als Typ von Domains und Spalten von Tabellen (table columns)

```
CREATE TABLE Pers    (Stammdaten    PersonT,
                      Lebenslauf    CLOB (75K),
                      Bild          BLOB (12M));
```

- Manipulation von UDT-Instanzen durch Aufruf von UDT-Funktionen
- Funktionsaufruf kann an jeder Stelle auftreten, wo ein skalarer Wert in SQL möglich ist
  - innerhalb von INSERT, UPDATE, DELETE, SELECT, ...
  - geschachtelte Funktionsaufrufe (Pfadausdrücke) sind möglich

```
INSERT INTO Pers VALUES ( PersonT ("Peter Schulz", Adresse ("Seestraße 12", "50321", "Köln")),
                                                    NULL, NULL)

SELECT Stammdaten.Name
FROM Pers
WHERE Stammdaten.Grundgehalt < 30000 AND contains (Lebenslauf, "Diplom")
```

REF-Typen

- dienen zur Realisierung von Beziehungen zwischen Typen bzw. Tupeln (OID-Semantik)

  <reference type> ::= **REF (** <user-defined type> **)** [SCOPE <table name>]

- nur “typisierte” Tabellen (aus strukturierten UDT abgeleitet) können referenziert werden

- jedes Referenzattribut muß sich auf genau eine Tabelle beziehen (SCOPE-Klausel)

```
CREATE TYPE PersonT
        (PNR        INT,
         Name       VARCHAR (40),
         Abt        REF (AbteilungT),
         Manager    REF (PersonT),
         Anschrift  Adresse,
         ...
);
```

```
CREATE TABLE  Pers OF PersonT
        (PRIMARY KEY PNR,
         SCOPE FOR Manager IS Pers
        •••)
```

- nur Top-Level-Tupel in Tabellen können referenziert werden
  - Referenzwert bleibt während Lebenszeit des referenzierten Tupels unverändert
  - DBS-weit eindeutige Referenzwerte
  - keine Wiederverwendung von Referenzwerten

- Verwendung von Referenzen innerhalb von Pfadausdrücken (Dereferenzierung über ->)

```
SELECT P.Name FROM PERS P
WHERE P.Manager->Name = “Schmidt” AND P.Anschrift.Ort = “Leipzig”
```

DISTINCT-Typen

- dienen der Wiederverwendung schon definierter Datentypen
  - einfache UDT, welche auf einem vordefinierten Datentyp basieren
  - mit neuen Typnamen versehen
  - DISTINCT-Typen sind unterscheidbar von dem darunterliegenden (und verdeckten) Basis-Typ

```
CREATE TYPE USDollar AS REAL;
CREATE TYPE CanDollar AS REAL;

CREATE TABLE US_SALES (Custno INTEGER, Total USDollar, ...)
CREATE TABLE Can_SALES (Custno INTEGER, Total CanDollar, ...)

SELECT C.Custno FROM US_SALES U, Can_SALES C
WHERE C.Custno = U.Custno AND


CREATE TYPE Schuhgröße AS INTEGER;
CREATE TYPE IQTyp AS INTEGER;
```

- keine direkte Vergleichbarkeit mit Basisdatentyp (Namensäquivalenz)
- keine Vererbung

Tupel-Typen (ROW-Typen)

- Tupel-Datentyp (row type)
  - Sequenz von Feldern (fields), bestehend aus Feldname und Datentyp
  - eingebettet innerhalb von Typ- bzw. Tabellendefinitionen

- Beispiel

```
CREATE TABLE  Pers
        (PNR          int,
        Name          ROW ( VName VARCHAR (20),
                            NName VARCHAR (20)),
        Anschrift     ROW (Strasse  VARCHAR (40),
                           PLZ      CHAR (5),
                           Ort      VARCHAR (40) )
        •••)
```

Kollektionstypen

- Kollektionstypen: Kollektionen (Set, List, Array ...) von Elementen gleichen Typs
- in SQL:1999 nur ARRAY vorgesehen

```
CREATE TABLE Umsatz
        (Produkt        CHAR (20),
        Monatsbetrag    REAL ARRAY [12],
        ...)
```

- Array-Operationen:
  - Element-Zugriff
  - Bildung von Sub-Arrays
  - Konkatenation (||) von Arrays

Generalisierung / Spezialisierung

- Spezialisierung in Form von Subtypen und Subtabellen
- in SQL:1999 nur Einfachvererbung
- es muß keine alleinige Wurzel in der Generalisierungshierarchie geben
- Subtyp
  - erbt alle Attribute und Methoden des Supertyps
  - kann eigene zusätzliche Attribute und Methoden besitzen
  - Methoden von Supertypen können überladen werden (Overriding)

CREATE TYPE PersonT (PNR INT, Name CHAR (20), Grundgehalt REAL, ...)
CREATE TYPE Techn-AngT **UNDER** PersonT (Techn-Zulage REAL, ...)
CREATE TYPE Verw-AngT **UNDER** PersonT (Verw-Zulage REAL, ...)

- Subtabellen (Teilmengenbildung) analog zu Typhierarchien

CREATE TABLE Pers OF PersonT (PRIMARY KEY PNR)
CREATE TABLE Techn-Ang OF Techn_AngT **UNDER** Pers
CREATE TABLE Verw-Ang OF Verw-AngT **UNDER** Pers

Substituierbarkeit

- Instanz eines Subtyps kann in jedem Kontext genutzt werden, wo Instanz eines Supertyps nutzbar ist
  - Eingabeargumente für Funktionen und Prozeduren, deren formale Parameter auf dem Supertyp definiert sind
  - Rückgabe als Funktionsergebnis, für das Supertyp als formaler Typ definiert wurde
  - Zuweisungen zu Variablen oder Attributen des Supertyps

- Verwendung von Supertypen in Tabellen kann somit zu heterogenen Tupelmengen führen

```
CREATE TABLE Pers OF PersonT
  ( ...)
```

```
INSERT INTO Pers VALUES (8217, 'Hans', 89500 ...)
INSERT INTO Techn_Ang VALUES (PersonT (5581, 'Rita', ...), 4300)
INSERT INTO Verw_Ang VALUES (PersonT (3375, 'Anna', ...), 5400)
```

Tabelle: *Pers*

PNR	Name	Techn-Zulage	Verw-Zulage
8217	Hans ...		
5581	Rita ...	4300	
3375	Anna ...		5400
1463	Elke ...		
...			

- Homogene Ergebnismengen mit ONLY-Prädikat bzw. durch Zugriff auf Subtabellen

```
SELECT *
FROM Pers ONLY
WHERE Grundgehalt > 80000
```

```
SELECT *
FROM Verw_Ang
WHERE Grundgehalt > 80000
```

Dynamisches Binden

- Overloading (Polymorphismus) von Funktionen und Methoden wird unterstützt

```
CREATE TYPE PersonT
  (PNR INT,  ... )
   METHOD Einkommen () RETURNS REAL,    ...
```

```
CREATE TYPE Techn_AngT UNDER PersonT
  (Techn-Zulage    REAL,  ...)
  OVERRIDING METHOD Einkommen ()
     RETURNS REAL,
  ...
```

```
CREATE TYPE Verw_AngT UNDER PersonT
  (Verw-Zulage     REAL,  ... )
   OVERRIDING METHOD Einkommen ()
     RETURNS REAL,
  ...
```

- Anwendungsbeispiel einer polymorphen Funktion:

```
CREATE TABLE Pers OF PersonT
  (...)
```

```
SELECT P.Einkommen
FROM Pers P
WHERE P.Name =’Müller’;
```

- dynamische Funktionsauswahl zur Laufzeit aufgrund spezifischem Typ

Rekursion

- Berechnung rekursiver Anfragen (z.B. transitive Hülle)

```
Create Table Eltern (    Kind        CHAR (20),
                         Elternteil  CHAR (20));

WITH RECURSIVE Vorfahren (K, V) AS
( Select * From Eltern Where Kind = "John"
  UNION
  Select E.Elternteil, V.V
       From Vorfahren V, Eltern E
       Where E.Elternteil = V.K)
Select V From Vorfahren
```

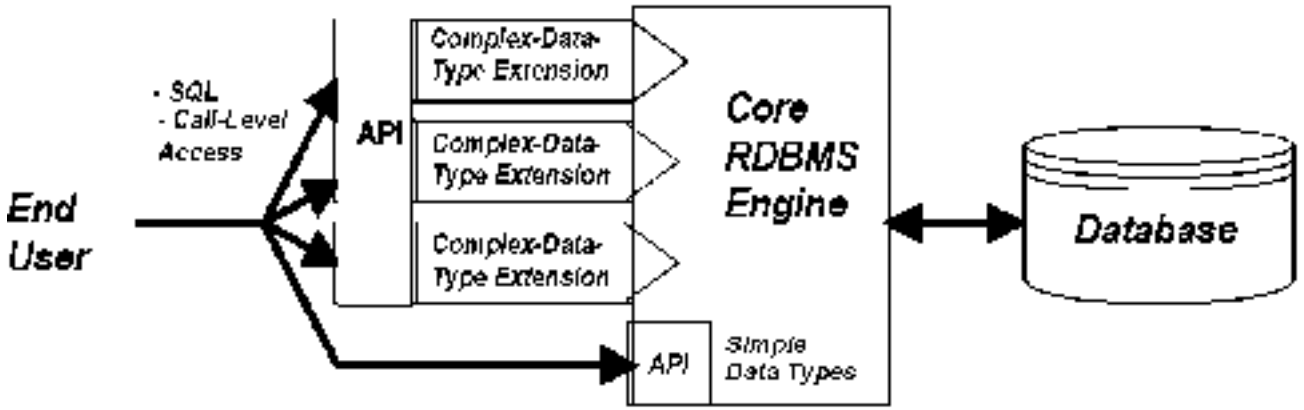
```
CREATE RECURSIVE VIEW Vorfahren (K, V) AS
( Select * From Eltern
  UNION
  Select E.Elternteil, V.V
     From Vorfahren V, Eltern E
     Where E.Elternteil = V.K)
```

- Syntax (WITH-Klausel)

```
<query expression> ::= [WITH [RECURSIVE] <with list> ] <query expression body>

<with list element> ::= <query name> [ ( <with column list> ) ] AS ( <query expression> )
                        [SEARCH <search order> SET <sequence column>] [CYCLE <cycle column list>]
<search order> ::= DEPTH FIRST BY <sort specification list> | BREADTH FIRST BY <sort specification list>
```

- Merkmale
  - verschiedene Suchstrategien (Depth First, Breadth First)
  - Zyklusbehandlung
  - lineare oder allgemeine Rekursion

Informix Universal Server



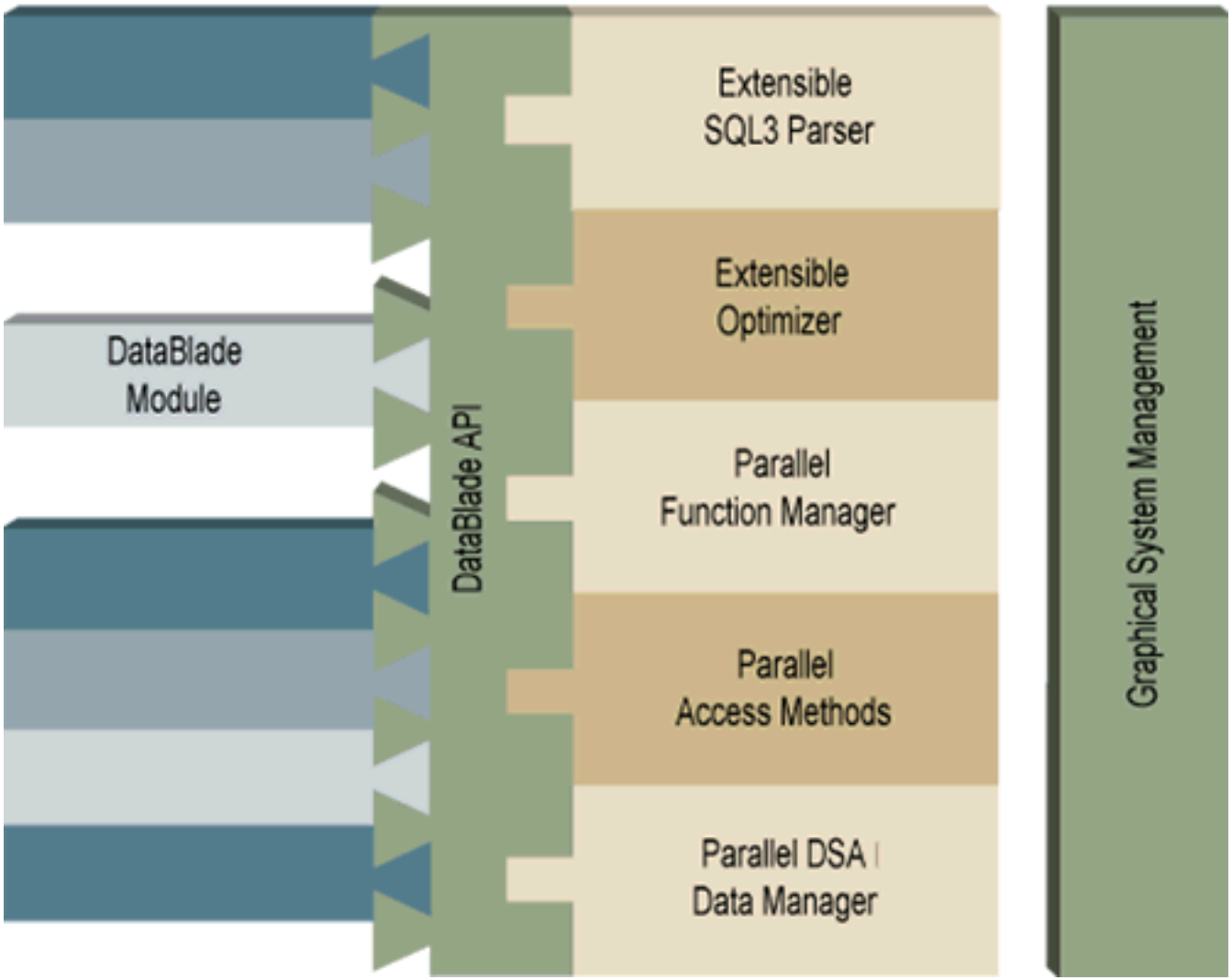
- Erweiterbarkeit über Data Blades (von Illustra übernommen)
  - benutzerdefinierte Datentypen
  - benutzerdefinierte Routinen
  - benutzerdefinierte Zugriffspfade (Indexstrukturen)
- Bisherige Data Blade Modules (insgesamt ca. 30)
  - 2D/3D Spatial
  - Image
  - Text
  - TimeSeries
  - Web • • •

Informix (2)

- viele SQL3-Features bereits realisiert
- umfassendere Kollektionstypen als in SQL:1999
- explizite Row-Typen (CREATE ROW TYPE Adresse ...)
- Anwendungsbeispiel

```
CREATE TABLE Pers
                (Name       VARCHAR (40),
                 Alter      INT,
                 Gehalt     INT,
                 Hobbies    SET (VARCHAR(25)),
                 Bild       IMAGE);


SELECT Name
FROM Pers
WHERE Bart (Bild) > 0.7 AND Alter > 60
```

Data Blades


DataBlade
Module
DataBlade API
Extensible
SQL3 Parser
Extensible
Optimizer
Parallel
Function Manager
Parallel
Access Methods
Parallel DSA
Data Manager
Graphical System Management

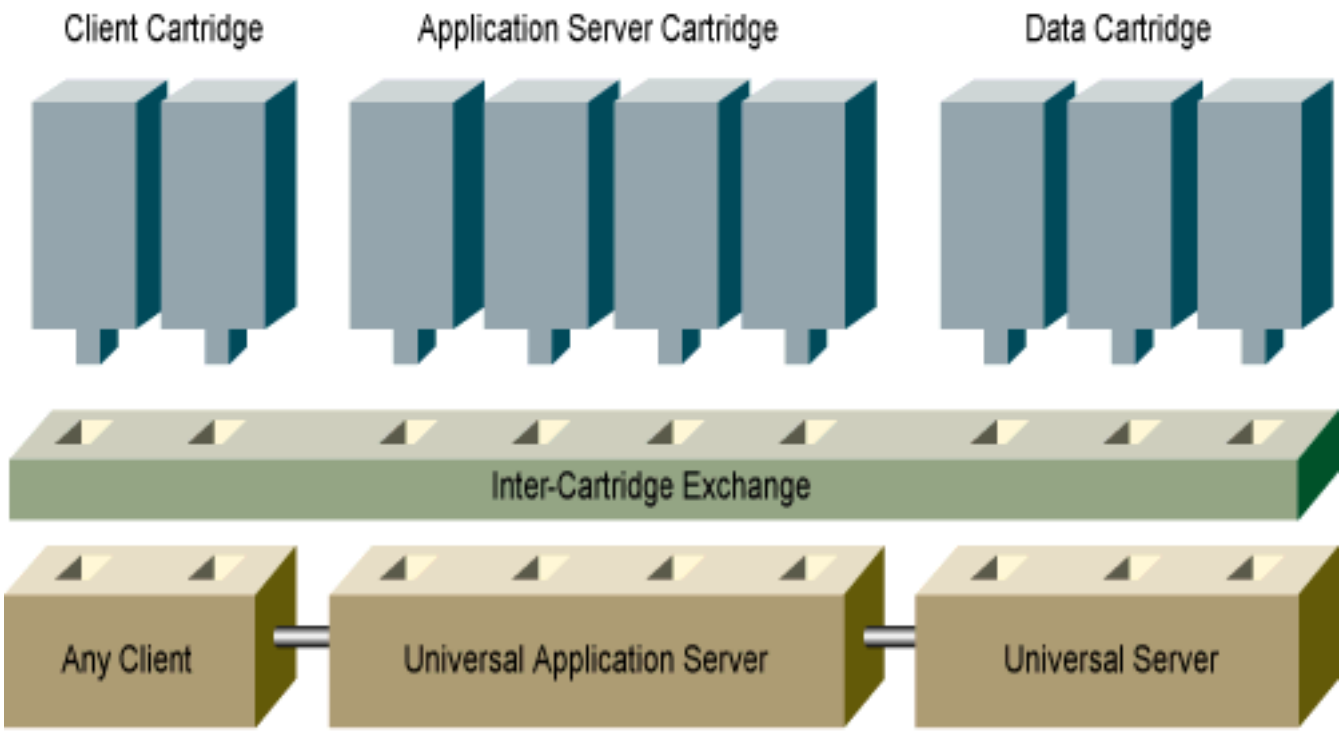
Text DataBlades

- Verwaltung von Dokumenten im Volltext
- Realisierung von Information Retrieval-Funktionen
  - Ähnlichkeitssuche
  - Thesaurusnutzung
  - Nachbarschaftssuche
  - Relevanzbewertung und Ranking von Ergebnissen
- Unterstützung unterschiedlicher Dokumentenformate (Word, PDF, HTML, SGML, ...)
- unterschiedliche Text DataBlades: Verity, OpenText, Excalibur, Arbor, IsoQuest
- Beispiel (Verity)

```
CREATE TABLE Dokumente (
                    DName VARCHAR (25),
                    DText DOC);


SELECT Dname, VTSRANK (DText, “Informix”)::int
FROM Dokumente
ORDER BY 2 DESC
```

Oracle

- objekt-relationale Features seit Oracle8
- Erweiterbarkeit über Data Cartridges (weniger eng integriert als Informix Data Blades)



- Unterstützung benutzerdefinierter Typen, Arrays sowie von (einfach) geschachtelten Tabellen
- keine Vererbung

Oracle (2)

- Geschachtelte Tabellen
  - Schachtelung auf typisierte Tabellen (“Objekt-Tabellen”) und maximal eine Schachtelungsstufe beschränkt
  - Zugriff über TABLE-Operator

- Beispiel

```
CREATE TYPE PersonT AS OBJECT (
                        PNR             INTEGER,
                        Name            VARCHAR2 (40),
                        Gebdatum        DATE, ...
                        MEMBER FUNCTION Alter RETURN INTEGER ... );

CREATE TYPE PersTabellenTyp AS TABLE OF PersonT;

CREATE TABLE Abt (
            ANR             INTEGER,
            Angestellte     PersTabellenTyp)
            NESTED TABLE Angestellte STORE AS AngTabelle;

INSERT INTO Abt VALUES (1, PersTabellenTyp (  PersonT (4711, “Müller”, ...),
                                              PersonT (4712, “Schulz”, ...), ...)
                        ... )

SELECT P.Name
FROM TABLE (SELECT A.Angestellte FROM Ant A WHERE A.ANR=1) P
WHERE P.Alter < 30
```

Zusammenfassung

- Objekt-relationale DBS
  - Beispiele: Informix Universal Server, DB2 Common Server, Oracle 8, UniSQL ...
  - Entwicklung steht noch am Anfang
  - zum Teil noch begrenzte Unterstützung der Objekt-Orientierung (meist keine Vererbung)
  - Nutzung nicht-standardisierter Erweiterungen
- OODBS
  - haben noch Entwicklungsvorsprung
  - optimale Performanz für navigierende Zugriffe aufgrund enger PS-Integration
  - erste Wahl für mit objektorientierter Programmiersprache realisierte Anwendungen , welche eine persistente Datenverwaltung benötigen
- SQL3-Standardisierung
  - Kompatibilität mit existierenden SQL-Systemen
  - Objektorientierung: Benutzerdefinierte Datentypen und Methoden, Typhierarchien und Vererbung, Objekt-Identität (REF-Typen)
  - erweiterbares Typsystem stellt signifikante Verbesserung der Modellierungsfähigkeiten dar
  - beschränkte Unterstützung von Kollektionstypen (Array)
  - Zahlreiche weitere Fähigkeiten (Trigger, Rekursion, • • •)
- Sehr hohe Komplexität für Benutzer und DBS-Implementierung

7. Heterogene Datenbanken*

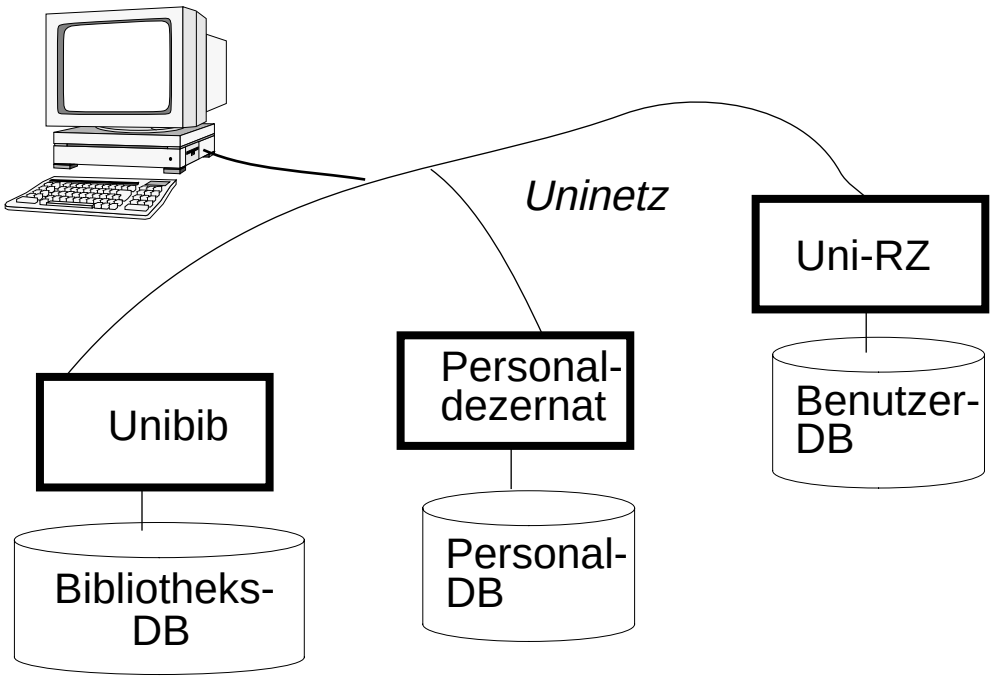
- Einführung
  - Autonomie vs. Heterogenität
  - Semantische Heterogenität
- Generelle Ansätze
  - Programmierte Verteilung vs. Verteilung von DB-Operationen vs. föderative DBS
  - Standardisierungen und Gateways
  - IBMs Distributed Relational Database Architecture (DRDA)
- Programmierte Verteilung
  - Verteilte Transaktionssysteme (TP-Monitore)
  - Verteilte Commit-Behandlung
  - X/Open DTP
- Verteilung von DB-Operationen: ODBC
- Föderative DBS
- Objektorientierte Ansätze (CORBA, OLE/DB)
- Beispiel-Realisierung: IBM DB2 DataJoiner

Data Warehousing -> Kap. 8

* E. Rahm: Mehrrechner-Datenbanksysteme, Addison-Wesley, 1994, Online-Version 1997 (MEDOC-Bibliothek)

Heterogene Datenbanken

- Inhaltlich verwandte Informationen einer Institution, eines Unternehmens, etc. sind in der Praxis häufig über mehrere heterogene Datenbanken verstreut, die unabhängig voneinander entworfen wurden und betrieben werden



- Heterogenität bezüglich
  - Hardware (Rechner, Peripherie, Kommunikationssystem, ...)
  - Betriebssystemen (Unix, MS/DOS, MVS, VMS, BS2000 ...)
  - Kommunikationsprotokollen (SNA, TCP/IP, Transdata, OSI ...)
  - DBVS (Hersteller, Version)
  - Datenmodellen (relational, objekt-or., CODASYL, hierarchisch)
  - Anfragesprache (SQL-Dialekt, DL/1, ...)
  - Transaktionsverwaltung (Synchronisation, Logging, Recovery)
  - Repräsentation der Daten (-> semantische Heterogenität)

Semantische Heterogenität

- v.a. durch Entwurfsautonomie eingeführt
- mögliche Behandlung durch Schemaintegration (=> föderative DBS)
- Namenskonflikte (Synonyme, Homonyme)
  - unterschiedliche Namen für dieselben Attribute/Relationen
  - derselbe Name für unterschiedliche Attribute/Relationen

  ⇒ Umbenennung
- Formatunterschiede (unterschiedliche Datentypen, Genauigkeit, etc.)

  ⇒ Einsatz von Konversionsfunktionen
- strukturelle Unterschiede
  - Repräsentation von Information durch Attribute vs. eigene Relation(en)
  - unterschiedliche Beziehungstypen (1:N, M:N, ...)
  - unterschiedliche Integritätsbedingungen (Eindeutigkeit, referentielle Integrität, Nullwertbehandlung, Defaultwerte, Wertebereiche, etc.) • • •
- Fehlende bzw. widersprüchliche Daten
  - Eingabefehler
  - unterschiedlicher Änderungsstand

Semantische Heterogenität: Beispiel

Datenbank 1 (UNIBIB):

PUBLIKATION (Pubnr, Titel, Typcode)

BUCHPUB (Pubnr, Verlag, Ejahr, #Exemplare, ISBN)

VERFASSER (Pubnr, Vname)

SCHLAGWORT (Pubnr, Sname)

Datenbank 2 (STADTBIB):

BUCH (ISBN, Titel, Autoren, Vnr, Jahr, Preis, Standort)

VERLAG (Vnr, Vname, Adresse)

Namenskonflikte

Strukturelle Konflikte

Datenkonflikte

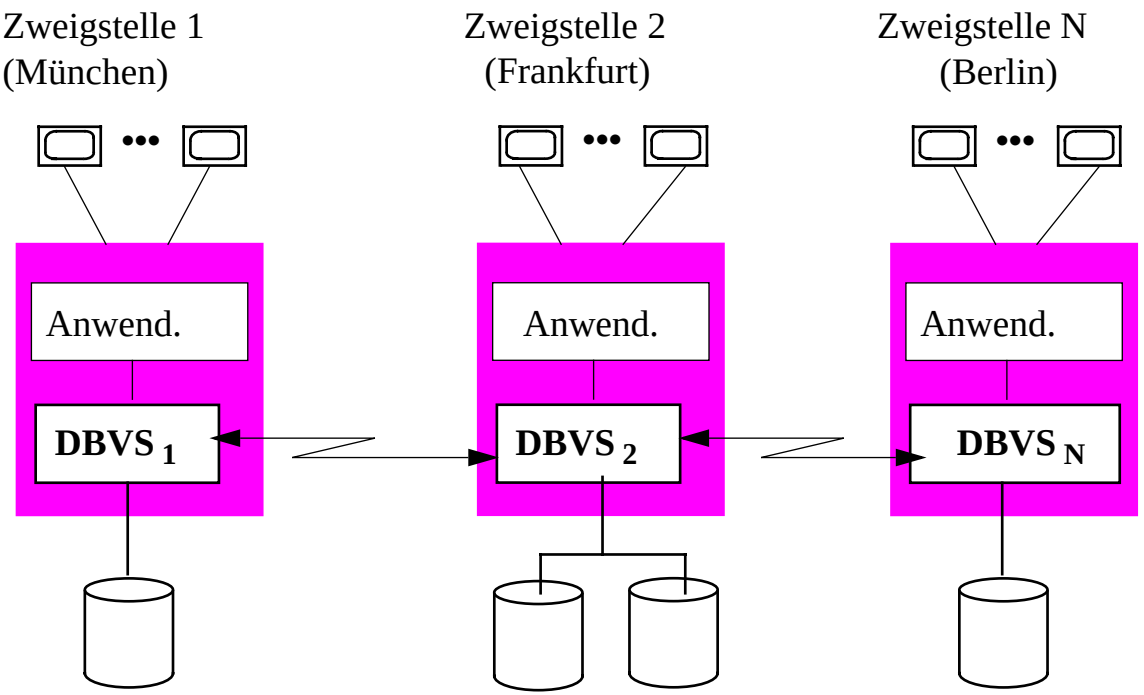
Heterogene Datenbanken (2)

- Anforderungen:
  - Zugriff auf mehrere Datenbanken innerhalb einer Transaktion (Wahrung der ACID-Eigenschaften)
  - einheitliche Zugriffsschnittstelle trotz Heterogenität bei den beteiligten DBS
  - Wahrung einer hohen Unabhängigkeit der einzelnen DBS (Knotenautonomie)
  - mächtige Zugriffsschnittstelle: in einer DB-Operation sollten Daten aus verschiedenen Datenbanken verknüpft werden können (Join-Bildung, etc.)
  - möglichst hohe Verteilungstransparenz

- Knotenautonomie (node autonomy)
  - Entwurfsautonomie: logischer DB-Entwurf, physischer DB-Entwurf, Wahl des lokalen DBS
  - Ausführungsautonomie
  - Kooperationsautonomie
  - Entwurfsautonomie Hauptursache für Heterogenität

- Grobe Alternativen zur Bearbeitung heterogener Datenbanken / Datenquellen
  - Vorabintegration der Datenquellen durch Verlagern in separate DB (-> Data Warehouse-Ansatz)
  - Beibehaltung selbständiger Datenbanken und Bereitstellung gemeinsamer Zugriffsmöglichkeiten

Verteilte DBS

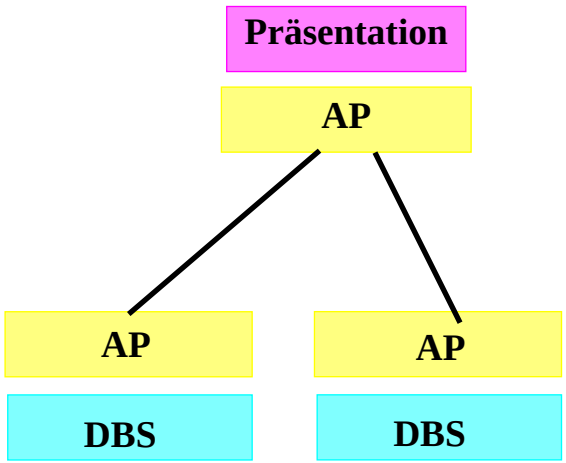


- Homogener und integrierter Ansatz: Verteilung 1 logischen DB unter mehrere Rechner (DBS)
- enge Kooperation zwischen DBS zur Gewährleistung von voller Verteilungstransparenz

  => ungeeignet zur Unterstützung heterogener Datenbanken

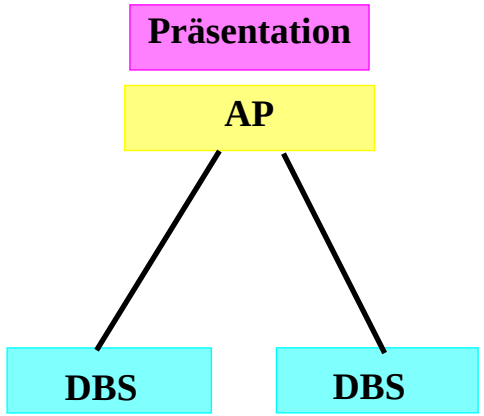
Generelle Alternativen für den Zugriff auf autonome DBS

Programmierte Verteilung



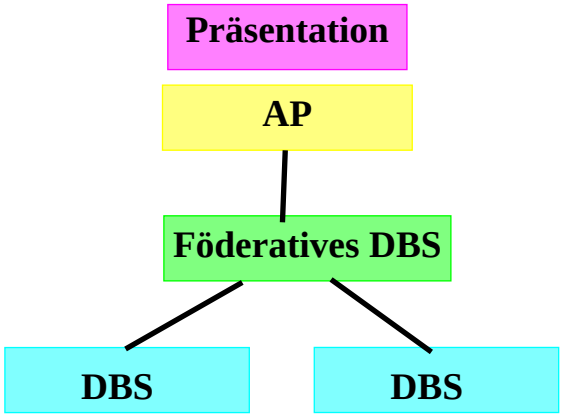
- DB-Zugriff über Aufruf vordefinierter Programmfragmente bzw. von Prozeduren/Methoden
- Realisierungsformen:
  - verteilte Transaktionssysteme (TP-Monitore)
  - verteilte OO-Systeme (Bsp.: CORBA)

Verteilung von DB-Operationen



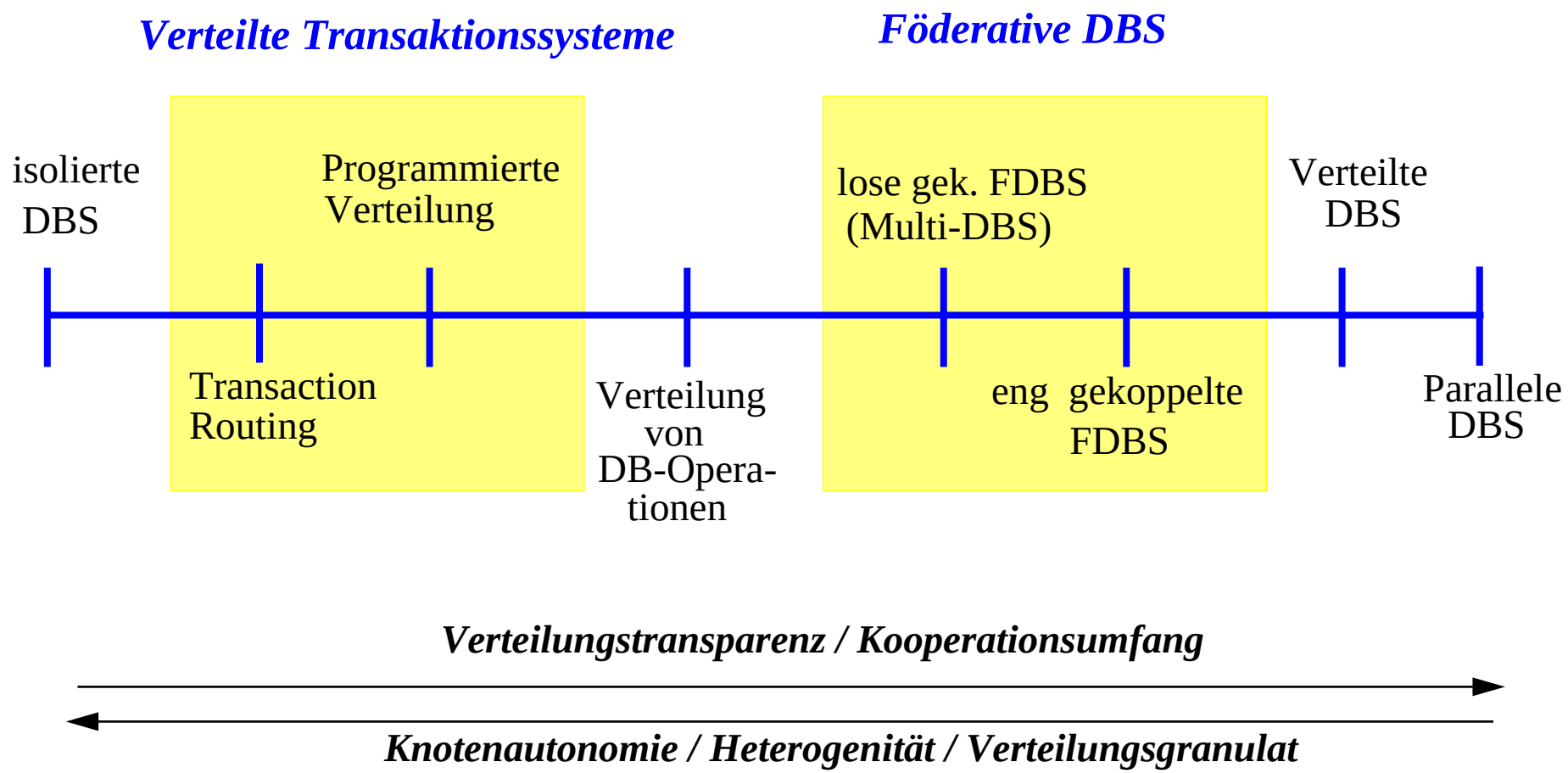
- Aufrufgranulat: DB-Operation (SQL-Anweisung)
- DB-Aufbau muß bekannt sein
- Client-DBS kann AP-Rolle übernehmen

Föderative DBS



- FDBS kann vereinheitlichte DB-Sicht unterstützen
- verteilte Ausführung einer DB-Operation

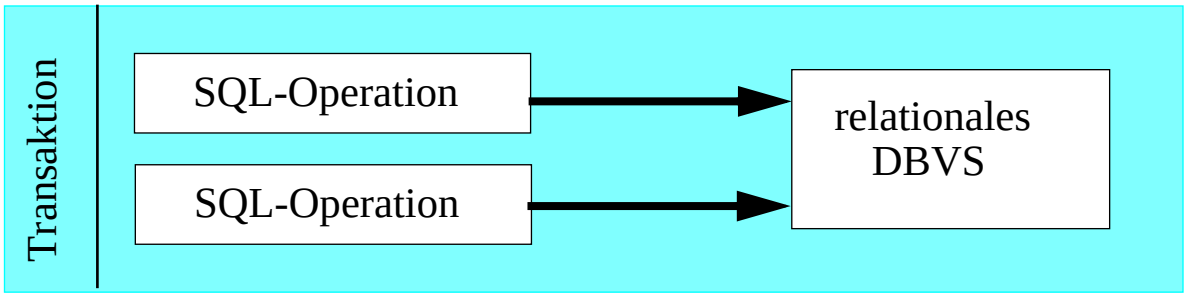
Kooperation vs. Autonomie/Heterogenität



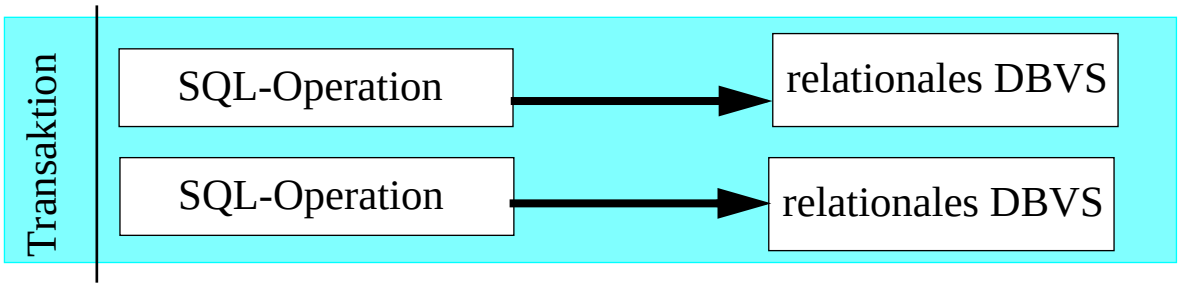
IBMs Distributed Relational Database Architecture (DRDA)

- Ziel: Interoperabilität zwischen SQL-DBS, v.a. von IBM (DB2/MVS, DB2/6000, DB2/2, DB2/400) sowie DRDA-kompatible Fremd-DBS
- Unterschiedliche Kooperationsformen

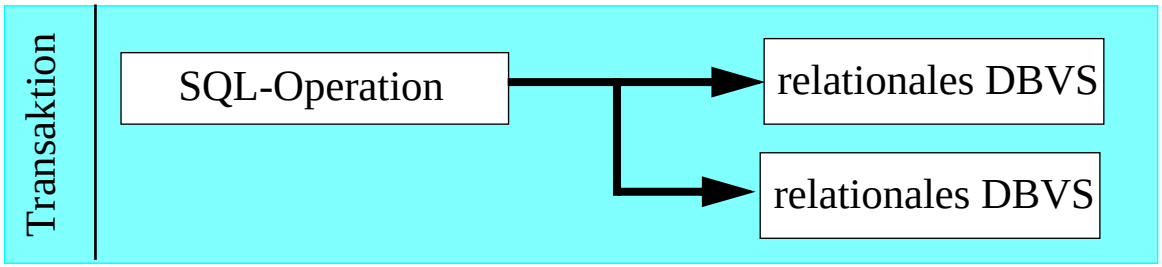
Remote Unit of Work



Distributed Unit of Work



Distributed Requests

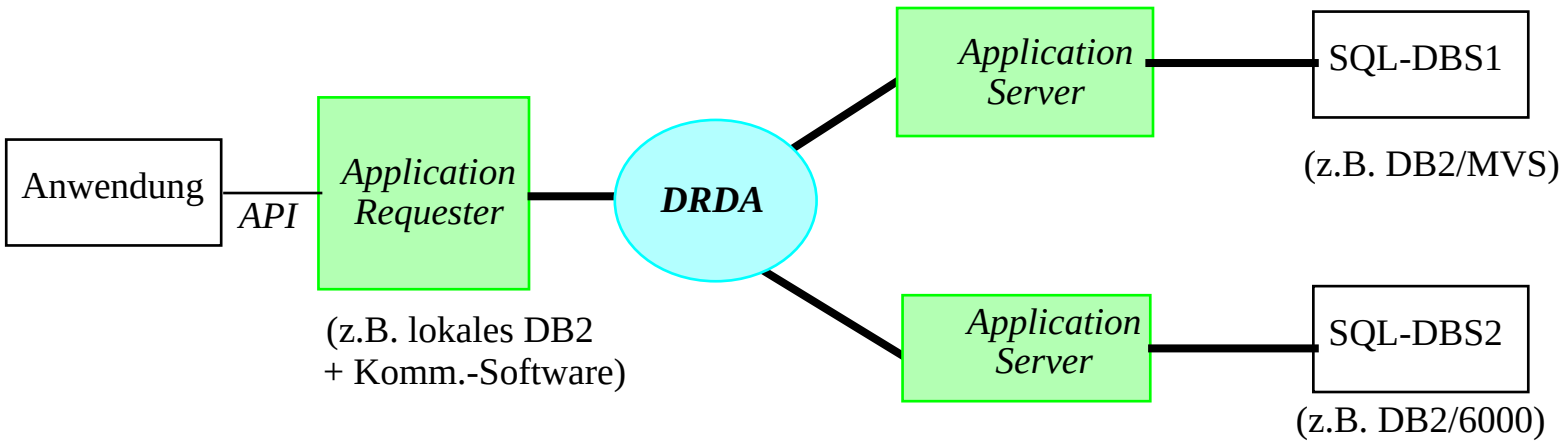


DRDA (2)

- Beispiel für “Remote Unit of Work”

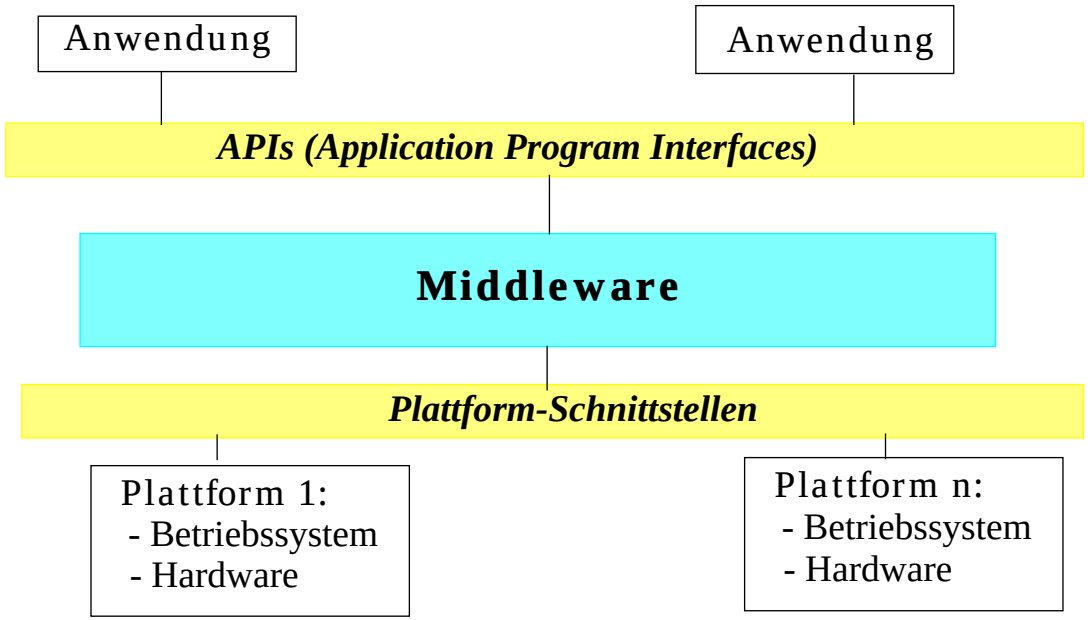
```
CONNECT TO L.DB2 ... ;
     SELECT ... FROM PERS1 ... ;
COMMIT WORK;
CONNECT TO F.DB2 ... ;
     SELECT ... FROM PERS2 ... ;
     UPDATE PERS2 SET ... ;
COMMIT WORK;
DISCONNECT ...
```

- Ablaufumgebung



- DRDA definiert Nachrichtenformate und -protokolle
- Abbildung SQL <-> DRDA-Nachrichten über Application Requester/Server
- verteilte Transaktionsverarbeitung auf Basis von LU6.2 (APPC)

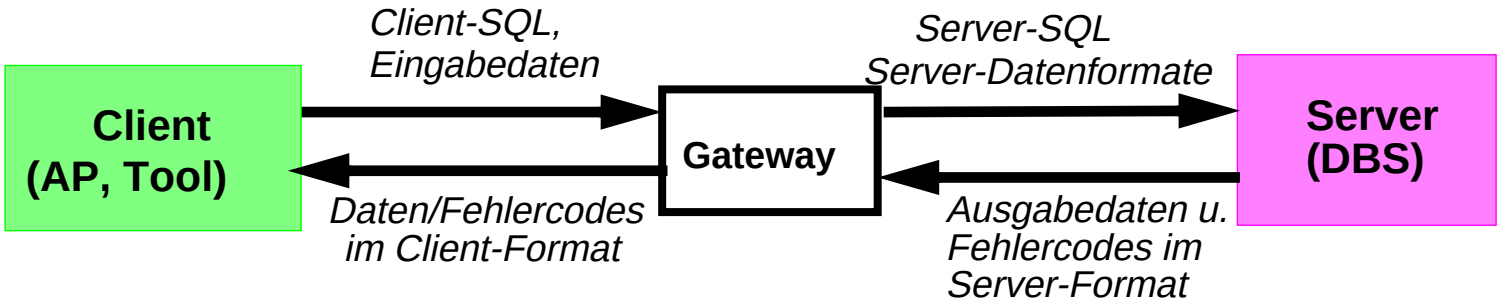
Notwendigkeit von Standards



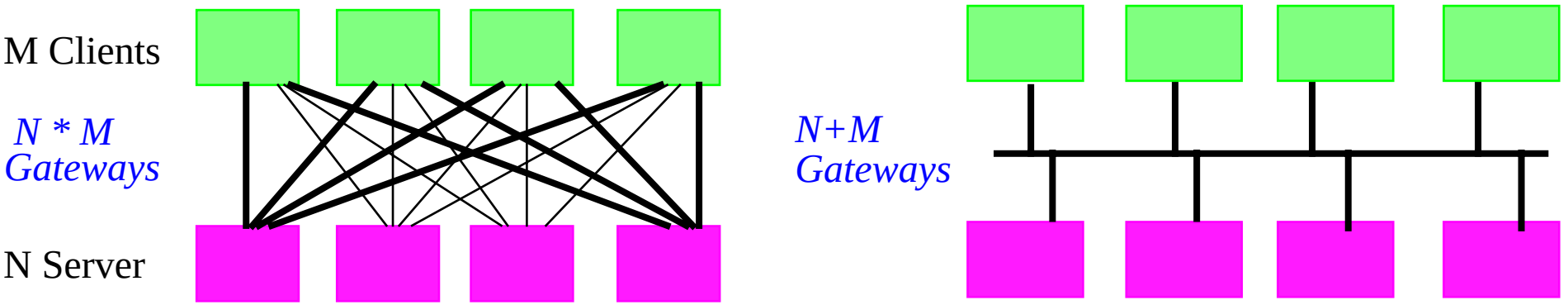
- Portabilität: standardisierte APIs
  - Funktionen für DB-Zugriff, Transaktionsverwaltung und Kommunikation
  - Teil-Standardisierung durch X/OPEN-Konsortium
  - Microsoft Open Database Connectivity (ODBC)
  - DB-Anfragesprache: ISO SQL bzw. normierter SQL-Subset
- Interoperabilität: standardisierte Kommunikation
  - Festlegung der Nachrichtenformate und Protokolle: *Formats and Protocols (FAP)*
  - Bsp.: ISO OSI, Teilprotokolle: RDA (Remote Database Access), TP, CCR, ROSE, ...
  - De-facto Standards: TCP/IP, LU6.2 (SNA)

Einsatz von DB-Gateways

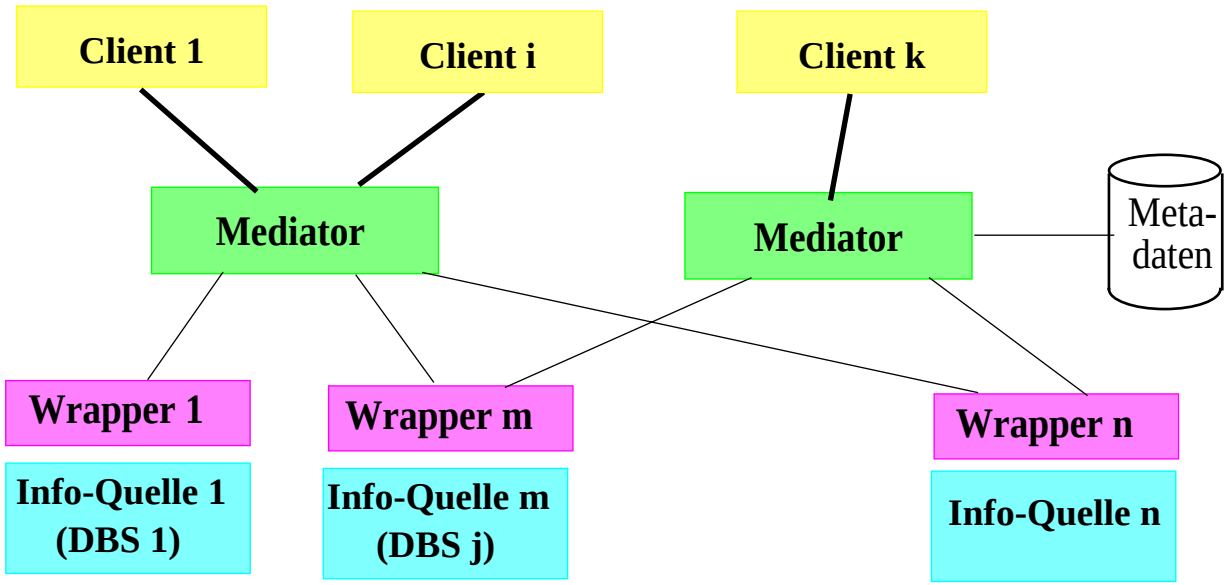
- selbst bei SQL-Datenbanksystemen erhebliche Unterschiede
  - Sprachumfang, Syntax, Datentypen, Fehlercodes, etc.
  - Gateways führen Transformationen zwischen Client- und Server-SQL durch



- hoher Aufwand zur Unterstützung zahlreicher Gateways (N*M)
- Definition einer gemeinsamen SQL-Teilmenge und einheitlicher Fehlercodes reduziert Gateway-Anzahl

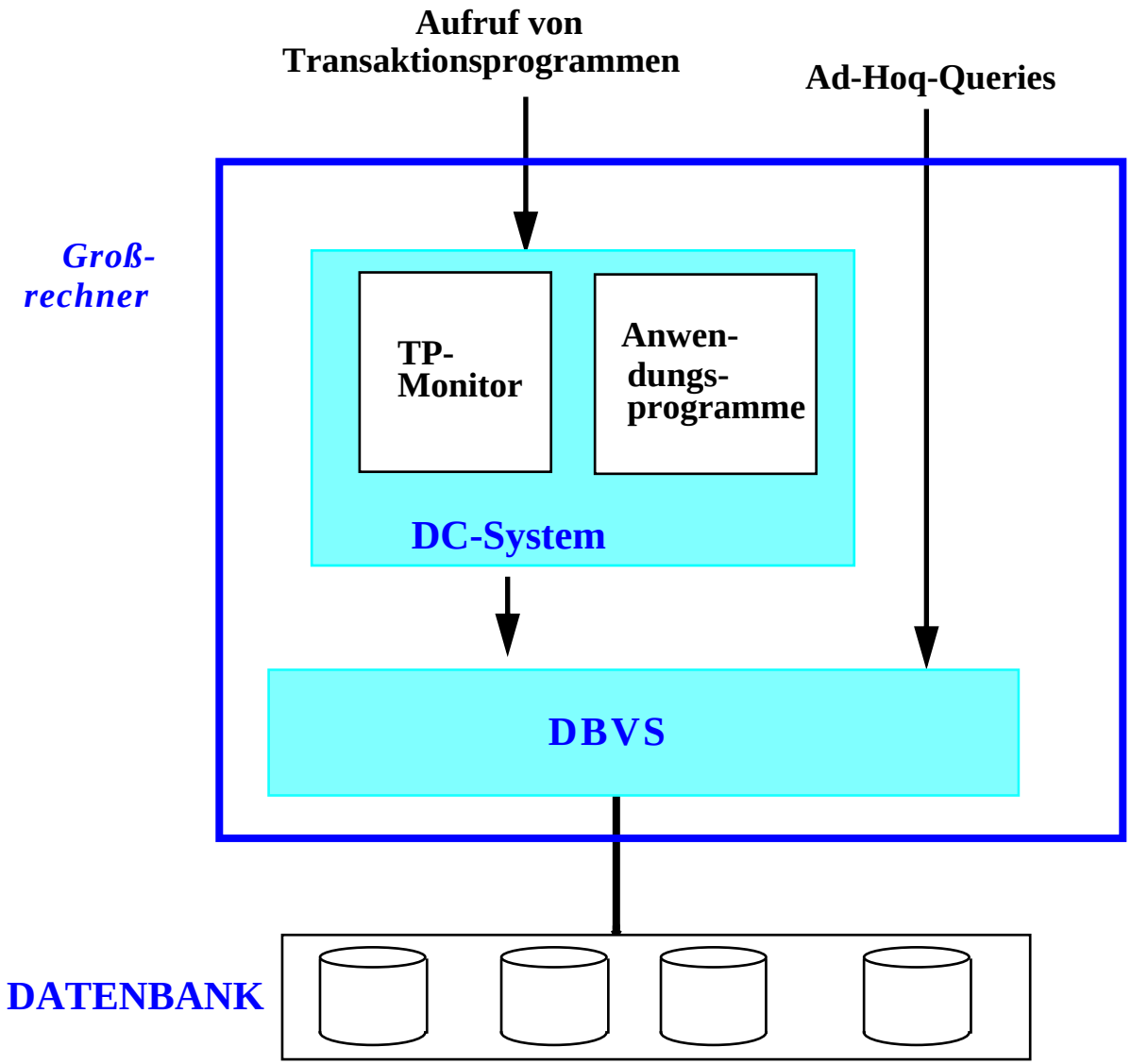


Einsatz von Mediatoren und Wrappern



- Mediator (Vermittler, Broker)
  - gemeinsame Zugriffsschnittstelle auf heterogene Datenquellen
  - Anfragezerlegung und Bündelung der Teilergebnisse
  - ggf. dynamische Bestimmung relevanter Informationsquellen
- Wrapper (Adapter)
  - Bereitstellung einer einheitlichen Zugriffsschnittstelle auf lokale Datenquelle
  - Gateway-ähnliche Funktionen zur Anpassung der Syntax, Ausgleich von Funktionsdefiziten etc.
  - Einbindung von unterschiedlichen DBS, Dateisystemen, Spezialsytemen etc.

Grobaufbau eines zentralisierten Transaktionssystemes



Entwurfsziele bei Transaktionssystemen

- Sicht des Endbenutzers
  - Aufruf von anwendungsspezifischen Funktionen (über Transaktionscode bzw. Selektion am Bildschirm)
  - Dateneingabe über vordefinierte Masken
    ⇒Konsequenz: Programm- und Datenunabhängigkeit

- Sicht des Anwendungsprogrammierers
  - Transaktionsprogramme (TAPs): Standardlösungen für immer wiederkehrende Aufgaben
  - Abstraktion für die TAPs: Datenzugriff und Kommunikation

DB-System

- logische Sicht auf die Daten
- Isolation der Programme
  - von den Zugriffspfaden und Speicherungsstrukturen *(Datenunabhängigkeit)*
  - von den Maßnahmen zur Sicherung und zum Schutz vor Wechselwirkungen *(Kontrollunabhängigkeit)*

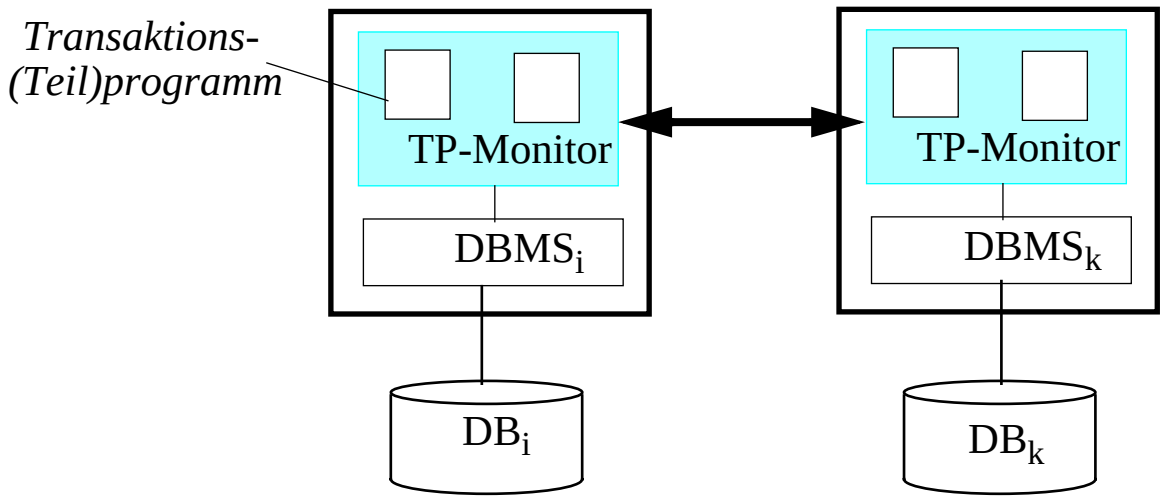
DC-System

- logische Sicht auf die Terminals
- Isolation der Programme
  - von den Kommunikationspfaden und Terminaleigenschaften *(Kommunikationsunabhängigkeit)*
  - von den Techniken des Multi-Tasking innerhalb eines Prozesses *(Kontrollunabhängigkeit)*

Verteilte Transaktionssysteme

- Verteilung unter Kontrolle des DC-Systems (TP-Monitor)
  - TP-Monitor verbirgt weitgehend Heterogenität bezüglich Kommunikationsprotokollen, Netzwerken, BS und Hardware
  - DBS bleiben weitgehend unabhängig (keine Kooperation zwischen DBS)
  - Verteilungsansatz: Programmierte Verteilung

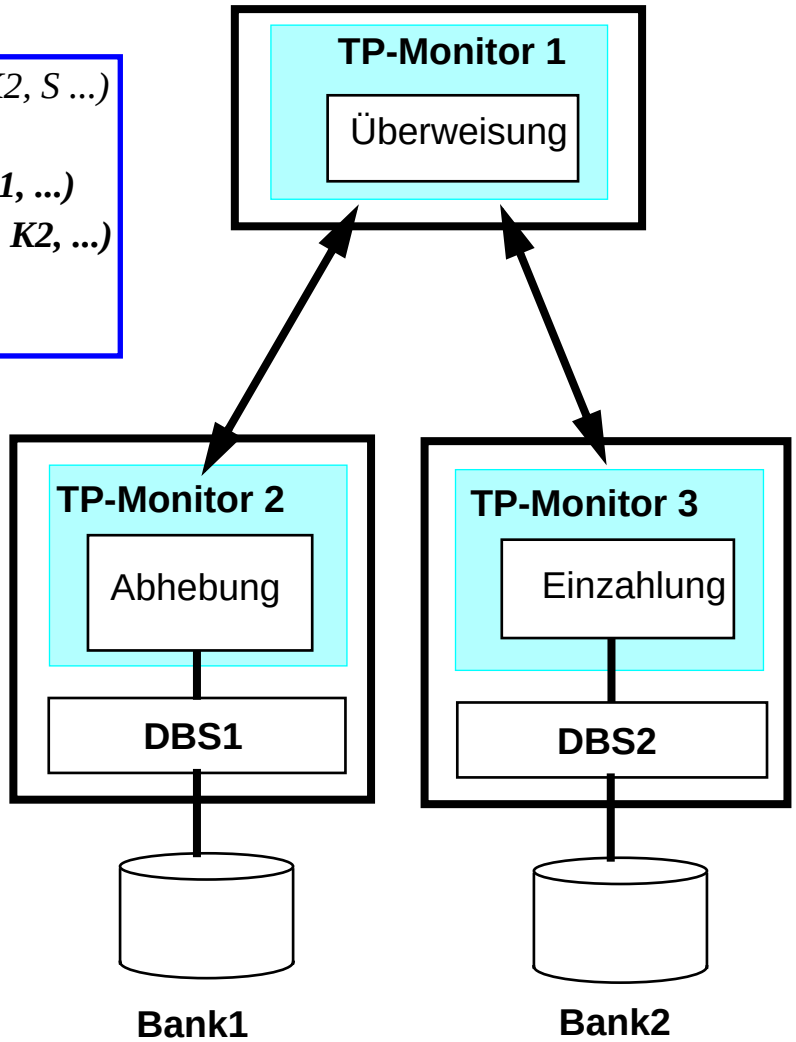
- Programmierte Verteilung
  - Verteilung auf Ebene von Anwendungsprogrammen
  - Zugriff auf externe DB durch Aufruf eines Teilprogrammes an dem betreffenden Rechner
  - jedes Teilprogramm greift nur auf lokale DB zu (mit jeweiliger Anfragesprache)
  - TP-Monitor unterstützt Kommunikation (RPC oder Peer-to-Peer) sowie Transaktionsverwaltung



Programmierte Verteilung: Beispiel

TAP Überweisung:

Eingaben (Parameter) lesen (K1, K2, S ...)
BEGIN TRANSACTION
CALL ABHEBUNG (Bank1, S, K1, ...)
CALL EINZAHLUNG (Bank2, S, K2, ...)
COMMIT TRANSACTION
Ausgabemitteilung



TAP Abhebung:

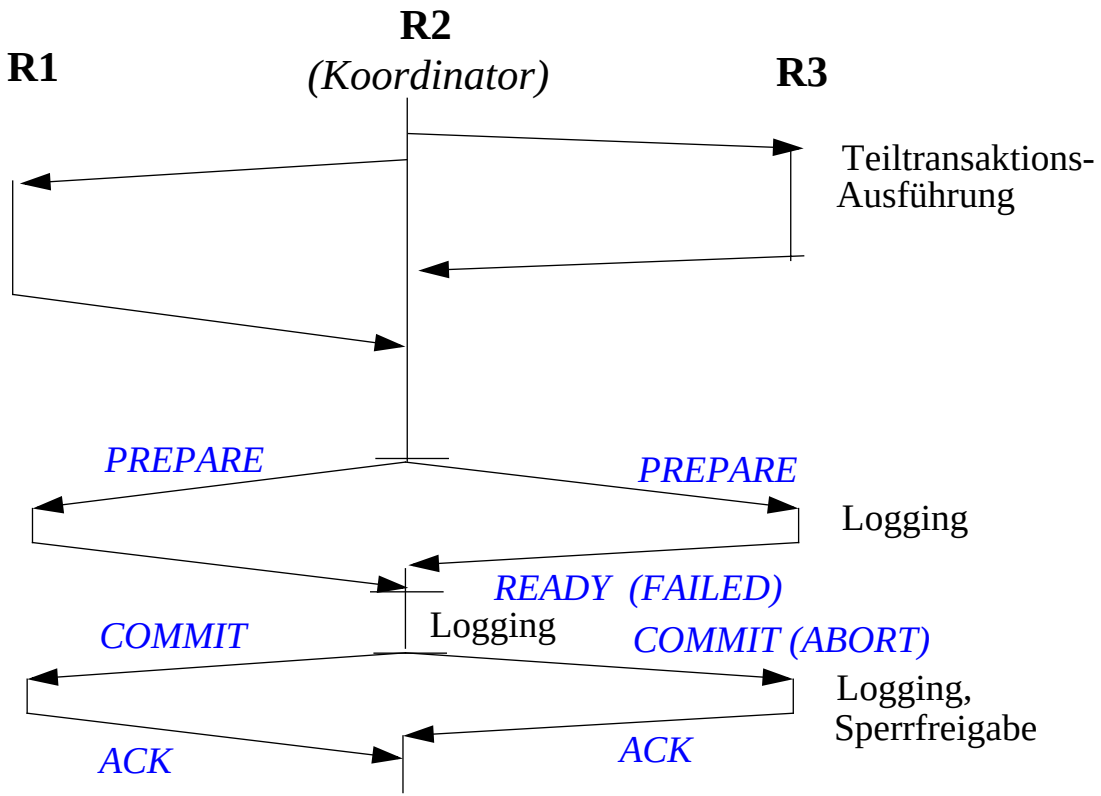
Parameter übernehmen
• • •
EXEC SQL UPDATE account
SET balance = balance - :S
WHERE acctno = :K1;
• • •
Ausführung zurückmelden

TAP Einzahlung:

Parameter übernehmen
• • •
EXEC SQL UPDATE GIROKTO
SET KSTAND = KSTAND+ :S
WHERE KNUMMER = :K2;
• • •
Ausführung zurückmelden

Verteiltes Commit-Protokoll

- Wahrung der Alles-oder-Nichts-Eigenschaft erfordert Durchführung eines verteilten Commits
- verbreitester Ansatz: verteiltes Zwei-Phasen-Commit mit zentralem Koordinator
  - Phase 1: Abstimmung über Commit bzw. Abbruch pro Rechner, an dem Transaktion aktiv war
  - Phase 2: Mitteilung des globalen Commit-Ergebnisses



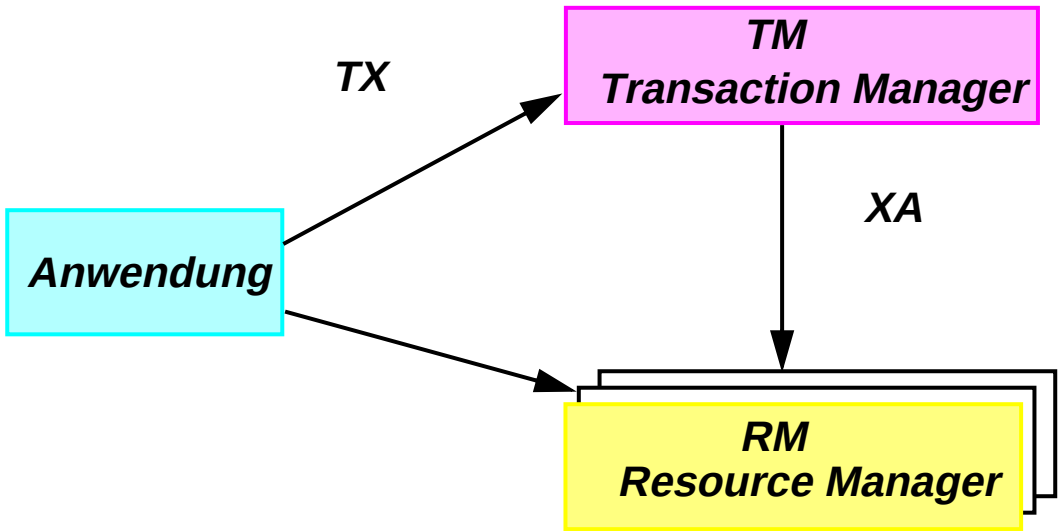
- Autonomiebeschränkungen: Teilnahme an festgelegtem Protokoll; Abhängigkeit zum Koordinator
- Koordinatorfunktion muß auf “sicherem” Rechner (Server-Rechner) ausgeführt werden

Verteilte Transaktionssysteme: Transaktionsverwaltung

- TP-Monitor koordiniert verteiltes Commit-Protokoll
- DBS indirekt beteiligt
  - müssen DB-Operationen (sowie Transaktions-ID) von nicht-lokalen Transaktionen akzeptieren
  - Teilnahme am Commit-Protokoll, um verteilte Änderungs-Transaktionen zu ermöglichen
- Commit-Protokoll gewährleistet nur die Eigenschaften A und D von ACID
- Isolation (I) erfordert globale Serialisierbarkeit
  - ist gewährleistet, falls jedes DBS ein striktes Zweiphasen-Sperrprotokoll (lange Lese- und Schreibsperren) zur Synchronisation verwendet
  - Auflösung globaler Deadlocks über Timeout

Verteilte Transaktionsverarbeitung nach X/Open (Distributed Transaction Processing, DTP)

- Definition eines allgemeinen Modells sowie von Schnittstellen zur verteilten Transaktionsverarbeitung in heterogenen Systemen



- TM koordiniert Transaktionsverwaltung (2PC)
- RM-Beispiele: DBS, Dateisysteme, Mail-Service, Window-Manager, ...
- RM müssen eigene Synchronisation, Logging und Recovery realisieren

- TX: Schnittstelle zur Transaktionsverwaltung
  - BEGIN, COMMIT, ROLLBACK von Transaktionen
  - Verbindungskontrolle zwischen Anwendungsprogramm und TM
- XA: Integrationsschnittstelle zwischen TM und DBS
  - zur Durchführung des 2-Phasen-Commit-Protokolls (2PC)
  - XA-kompatible DBS erlauben Commit-Initiierung von “außen”

X/OPEN DTP (2)

TX-Schnittstelle

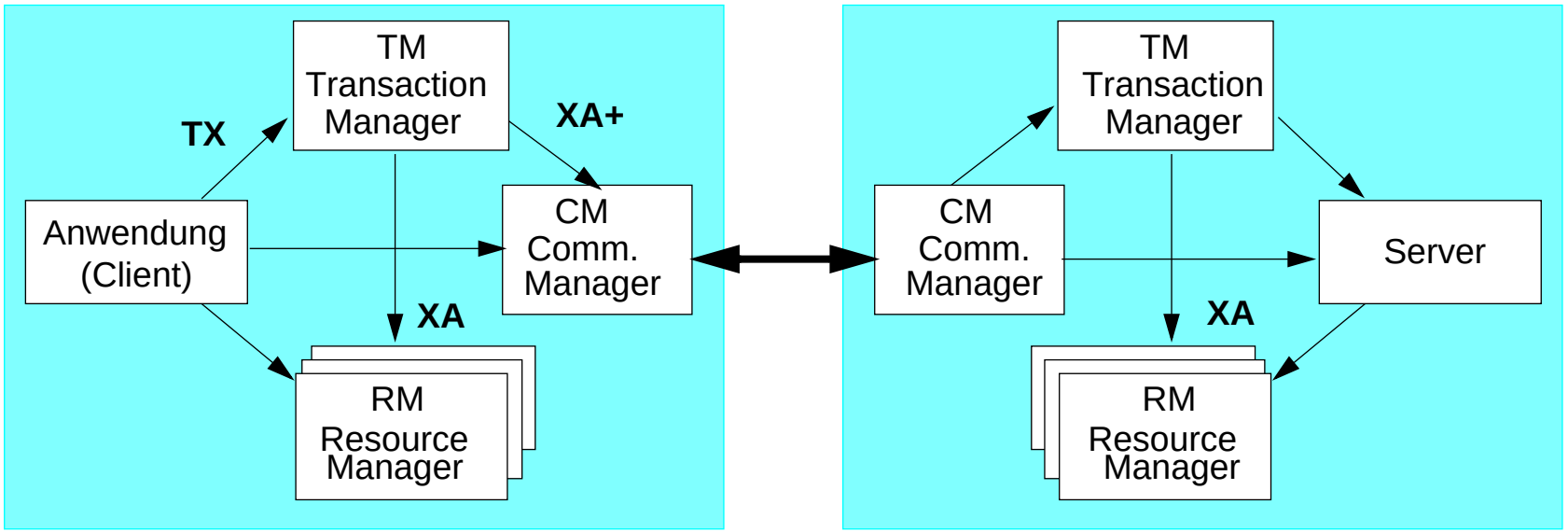
tx_open	Öffnen der dem TM bekannten RM
tx_close	Beenden der Verbindung zwischen AP und TM
tx_begin	Starten einer globalen Transaktion
tx_commit	erfolgreiches Beenden einer globalen Transaktion
tx_rollback	Rücksetzen einer globalen Transaktion
tx_info	Informationen über aktuelle Transaktion anfordern
tx_set_TA_control	“chained” Transaktions-Modus ein-/ausschalten
tx_set_TA_timeout	Timeout für automatisches Rollback setzen
tx_commit_return	Fortsetzen des AP schon nach Phase 1 des 2PC-Protokolls

XA-Schnittstelle

ax_reg	Registrieren eines RM beim TM
ax_unreg	Abmelden eines RM beim TM
xa_open	Initialisierung RM für AP
xa_close	RM-Nutzung beenden für AP
xa_start	RM nimmt an neuer Transaktion teil
xa_end	RM beendet Arbeit an Transaktion
xa_prepare	Aufforderung zur Commit-Vorbereitung
xa_commit	Commit-Aufforderung
xa_rollback	Rollback-Aufforderung
xa_complete	Nachfrage, ob xa-Aufruf beendet
xa_forget	RM kann Information über heuristisch beendete Transaktion vergessen
xa_recover	Anforderung von IDs zu Transaktionen, die im RM im Zustand “prepared” bzw. heurist. beendet sind

X/OPEN DTP (3)

- Kommunikation über Communication-Resource-Manager
  - standardisierte Kommunikation über RPC oder Peer-to-Peer
  - keine Verteilungstransparenz
- OSI TP als verteiltes Commit-Protokoll zwischen TM (XA+)

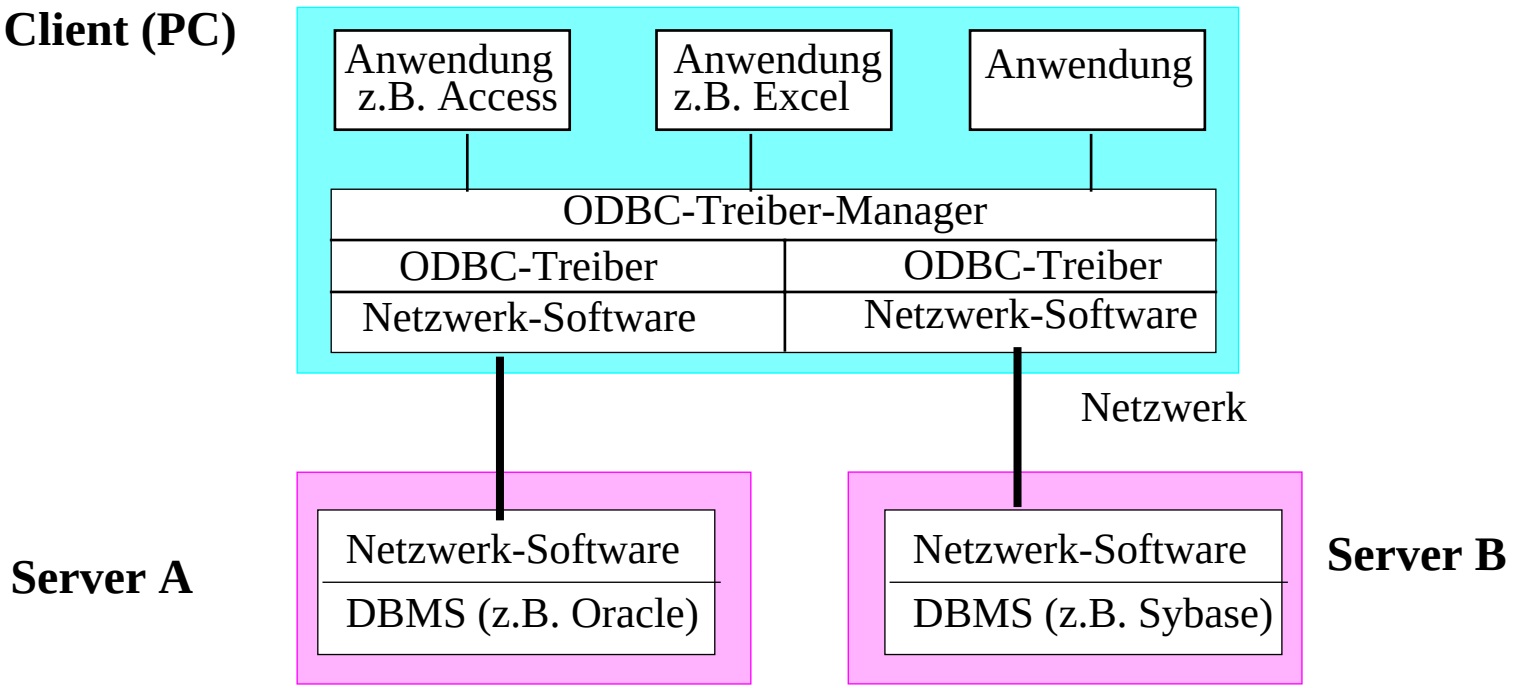


Vergleich

	Programmierte Verteilung	DRDA (Stufe 2)	ODBC	Föderative DBS	OLE / DB, CORBA	Data Warehouse
hohe Knotenautonomie für DBS						
einheitliche DB-Zugriffsschnittstelle						
verteilte Transaktionen						
verteilte DB-Operationen (z.B. Joins)						
hohe Verteilungstransparenz						

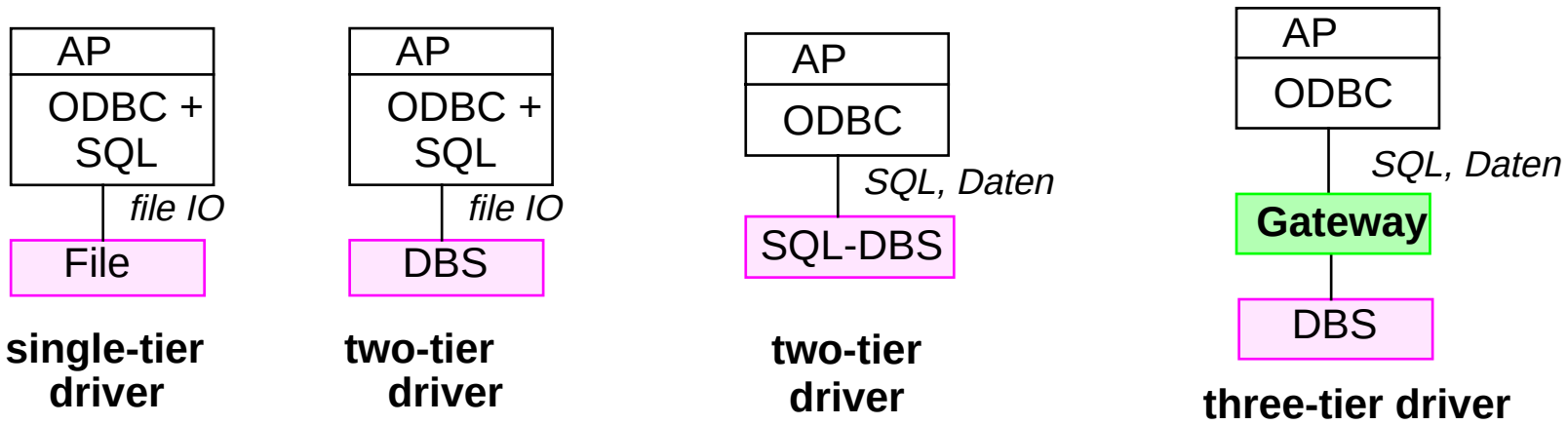
Open DataBase Connectivity (ODBC)

- API-Definition von Microsoft für einheitlichen Zugriff auf SQL-Datenbanken
- Call-Level-Interface; nur dynamisches SQL
- Verteilungsform: Verschicken von DB-Befehlen
- pro SQL-Server eigener “ODBC-Treiber”auf Client-Seite erforderlich
- Transaktionen sind auf 1 Server beschränkt



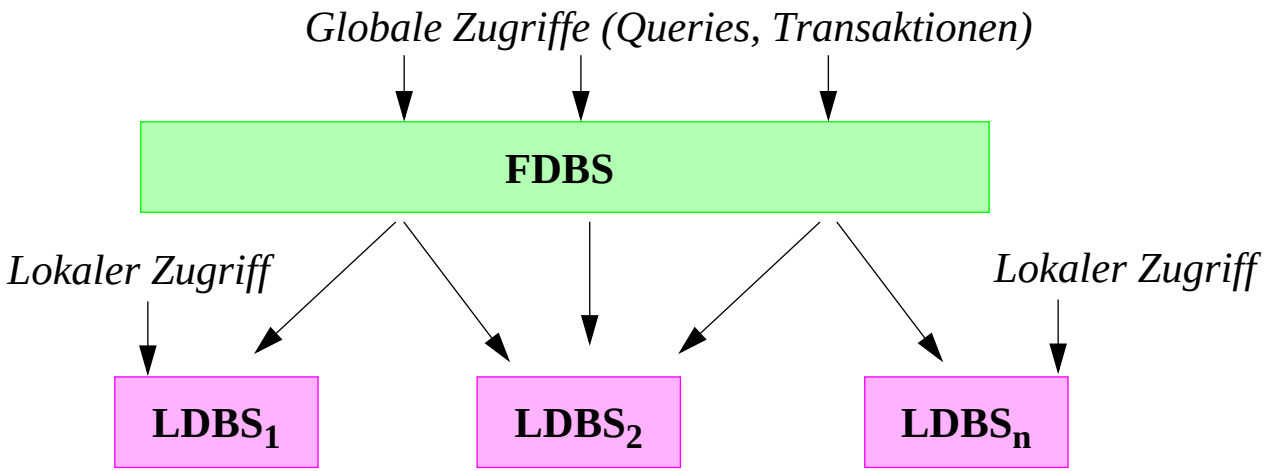
ODBC (2)

- unterschiedlicher Treiberumfang
  - Kernfunktionen und -Datentypen nach X/Open CLI
  - Erweiterungen (Level 1 oder Level 2): Date, Time, Scroll-Cursor, asynchrone Befehlsausführung etc.
- 1-stufige oder mehrstufige Verarbeitung von DB-Operationen
- 1-stufige Treiber (one-tiered driver) für Dateizugriffe
  - direkter Datenzugriff (lokal oder entfernt)
  - Treiber verarbeitet SQL-Anweisungen für Nicht-SQL-Systeme
  - Bsp.: Zugriff auf Xbase-Dateien
- mehrstufiger Treibereinsatz
  - SQL-Anweisungen werden an Server-DBS weitergegeben
  - Zugriff auch Nicht-SQL-DBS über SQL-Unterstützung auf Client- oder Server-Seite, ggf. über DB-Gateway (-> 3-stufiger Ansatz)

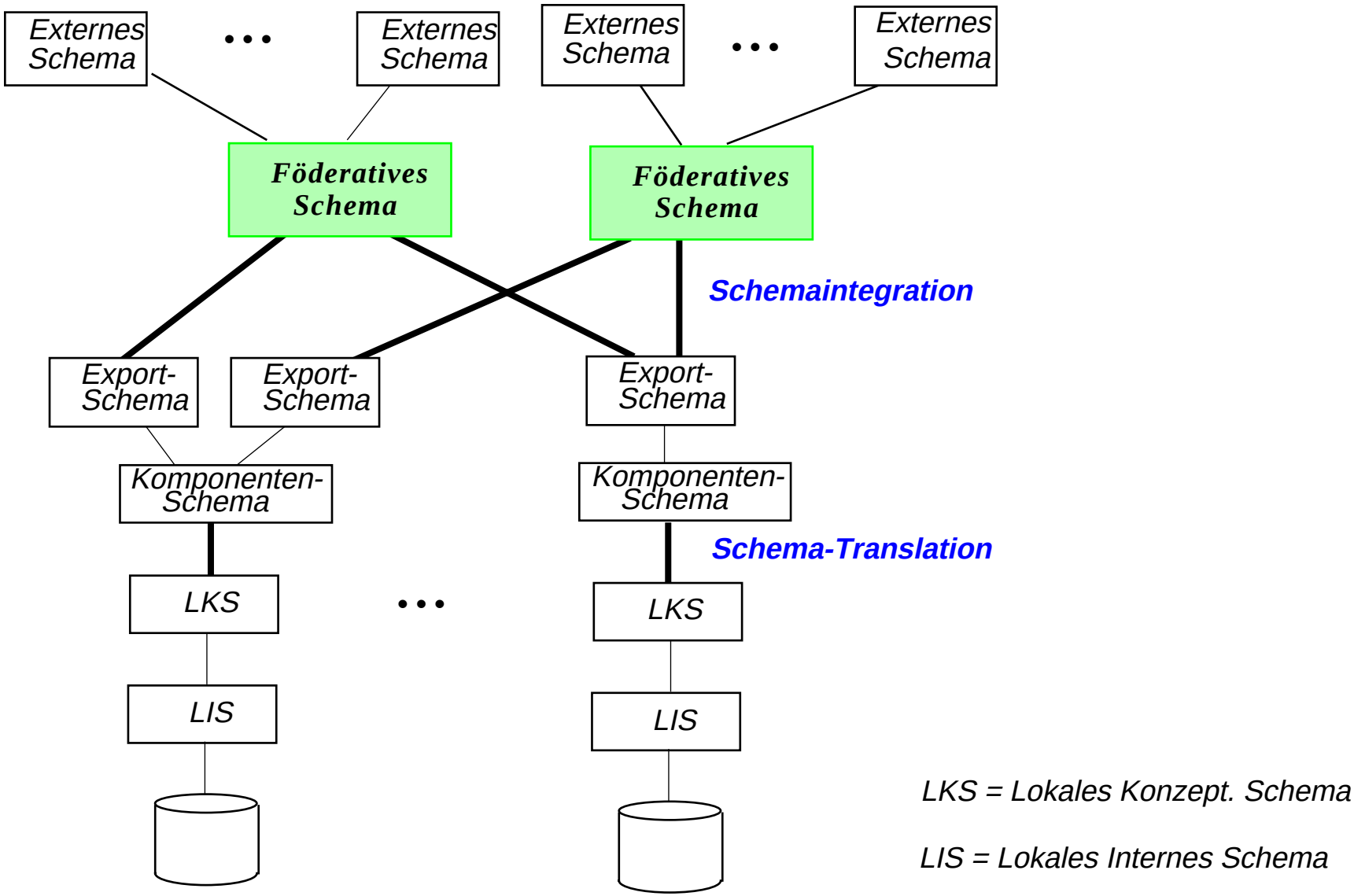


Föderative Mehrrechner-DBS (FDBS)

- Bildung von Föderationen zwischen existierenden, unabhängigen DBS zum Datenaustausch
  - begrenzte Kooperation zwischen DBS
  - Wahrung einer möglichst hohen Knotenautonmie
  - Unterstützung für heterogene DBS
- weitergehende Funktionalität als mit verteilten Transaktionssystemen
  - DBS-übergreifende Operationen
  - höhere Verteilungstransparenz
  - einheitliche Anfragesprache
  - Unterstützung bezüglich Heterogenität bei Datenmodellen und Transaktionsverwaltung
  - Unterstützung bezüglich semantischer Heterogenität
- Zusatzebenen-Architektur



Schemaarchitektur von FDBS



LKS = Lokales Konzept. Schema

LIS = Lokales Internes Schema

Schemaintegration

- Wesentliche Integrationschritte
  - Vorintegration
  - Erkennung von Namens- und strukturellen Konflikten
  - Auflösung der Konflikte (Conformation)
  - Mischen und Restrukturierung

- Vorintegration
  - Wahl der Integrationsstrategie (binär, n-stellig)
  - Reihenfolge unter den Schemata festlegen
  - Festlegung der Schlüsselkandidaten
  - Festlegung äquivalenter Wertebereiche sowie von Konversionsfunktionen zwischen ihnen

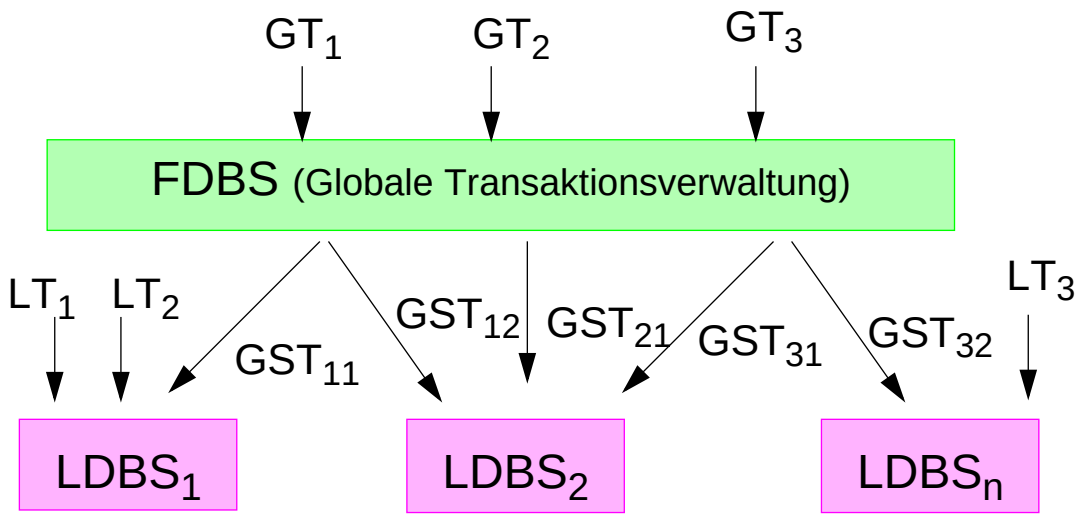
- Auflösung struktureller Konflikte oft schwierig
  - Schematransformationen (Attribute <-> Relationen, ...)

- Mischen und Restrukturierung auch kaum automatisierbar
  - zunächst Mischen der einzelnen Schemata
  - Restrukturierung zur Reduzierung von Redundanz
  - ggf. Transformationsfunktionen für Datenkonflikte

Schemaintegration: Bibliotheksbeispiel

FDBS: Transaktionsverwaltung

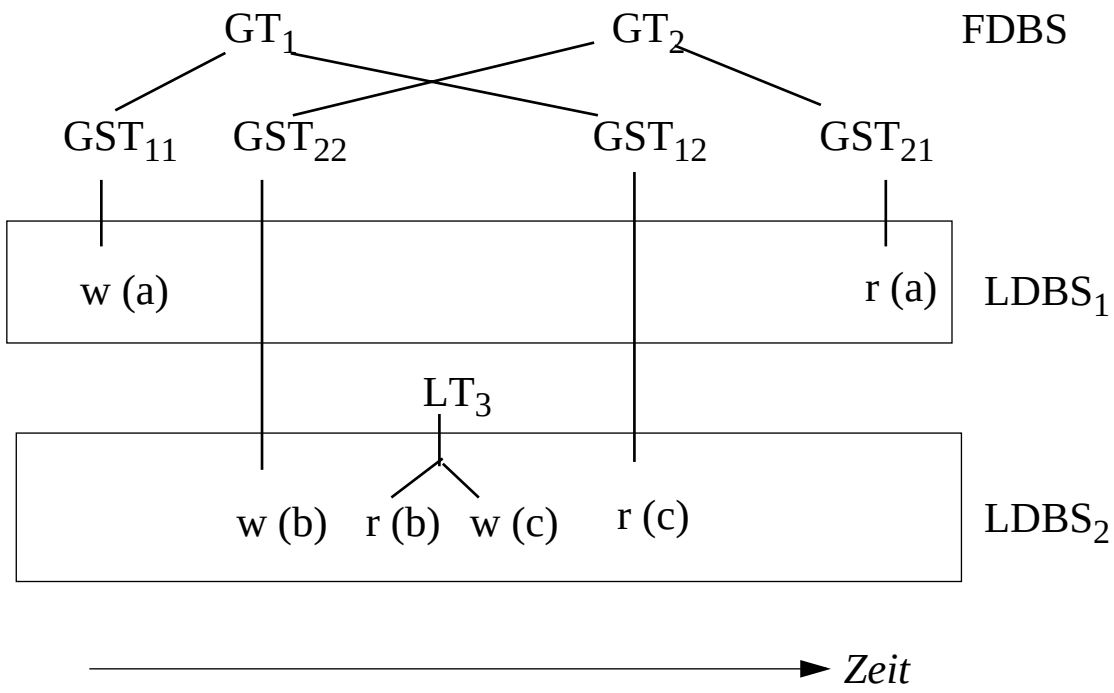
- zweischichtige Transaktionsverwaltung
  - lokale Transaktionen (LT)
  - systemübergreifende Transaktionen: Zerlegung in globale Subtransaktionen (GST) durch FDBS



- Lokale Autonomie impliziert
  - Kontrollinformation eines LDBS (für Synchronisation und Recovery) wird dem FDBS nicht verfügbar gemacht.
  - ein LDBS kann nicht zwischen lokalen und globalen Transaktionen unterscheiden
    => GST werden wie LT behandelt (ACID)
  - LDBS entscheiden unabhängig über das Commit lokal ausgeführter Transaktionen
  - bereits existierende lokale Programme werden nach der Integration weiterbenutzt
  - heterogene lokale Transaktionsverwaltung

FDBS: Synchronisation

- FDBS kann nur GTs synchronisieren
  - Annahme: alle Objekte, auf die zugegriffen wird, können durch FDBS identifiziert werden
  - Erkennung nur von direkten Konflikten zwischen GST_{ij} möglich
- Probleme durch transitive Abhängigkeiten mit lokalen TA



$\Rightarrow$ erkannte Abhängigkeiten in FDBS: $GT_1 \rightarrow GT_2$

FDBS-Transaktionsverwaltung: Lösungsansätze

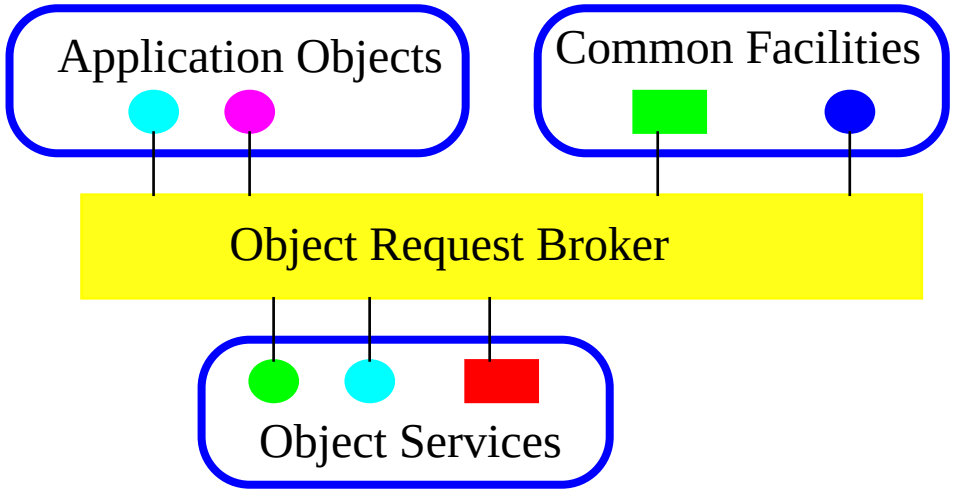
- Einschränkungen bezüglich
  - Funktionalität (z.B. begrenzte Unterstützung globaler Änderungstransaktionen)
  - LDBS-Autonomie (v.a. Erweiterungen der lokalen Transaktionsverwaltung)
  - Heterogenität (z.B. Verwendung identischer Synchronisations- und Commit-Protokolle in den LDBS)
  - ACID-Zusicherungen (z.B. Verzicht auf globale Serialisierbarkeit)

- Einfacher “Ansatz”: Beschränkung globaler Transaktionen auf höchstens eine ändernde Sub-Transaktion

- X/Open-Ansatz:
  - standardisiertes Commit-Protokoll (XA-Schnittstelle)
  - Striktes 2PL in jedem LDBS
  - Timeout zur globalen Deadlock-Behandlung

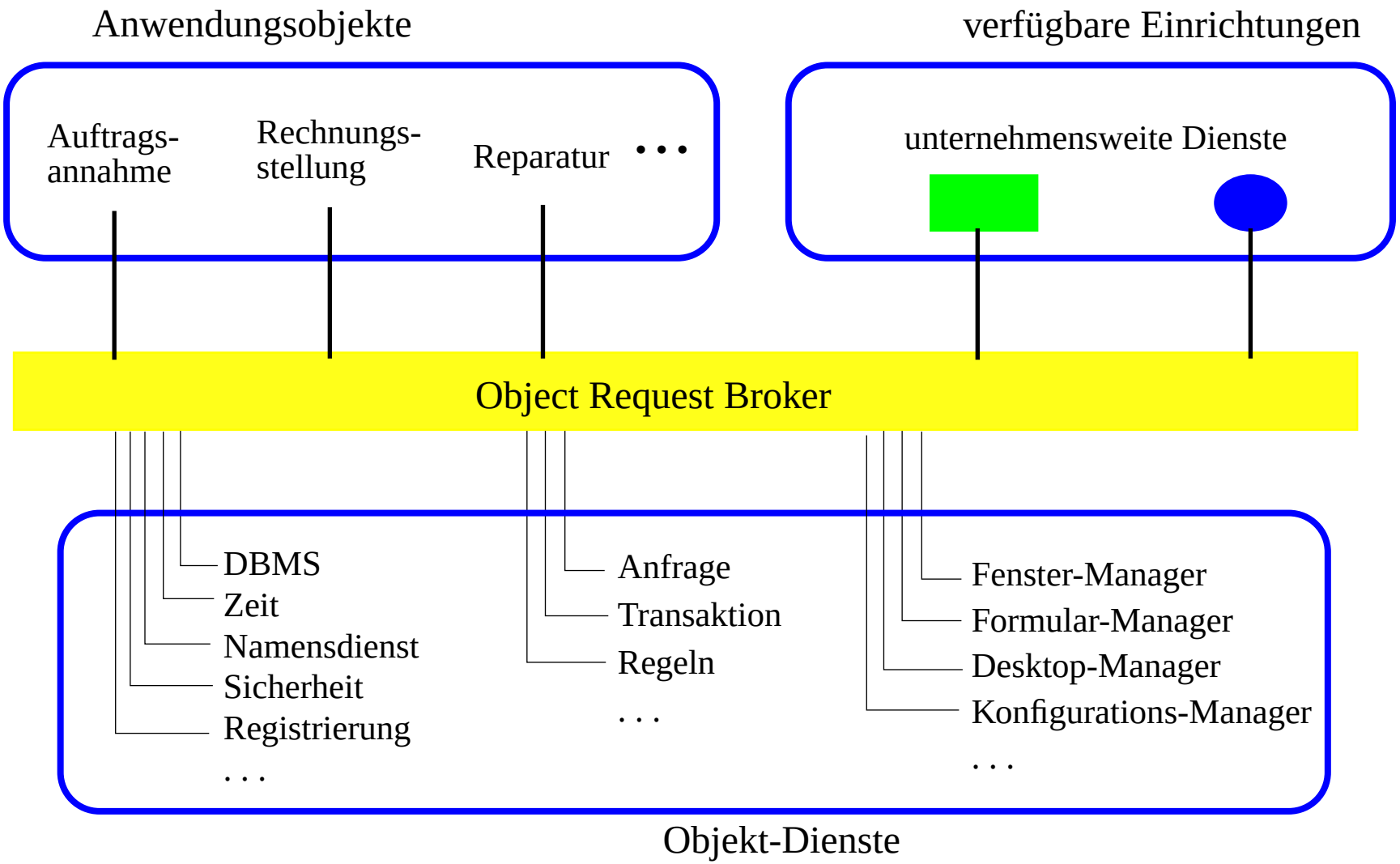
- Forschung: zahlreiche Vorschläge v.a. zur Synchronisation
  - Ziel: höherer Grad an Autonomie und Heterogenität als beim X/Open-Ansatz
  - oft jedoch fragliche Annahmen: zentralisierter globaler Transaktions-Manager, Kenntnis der Referenzen globaler Transaktionen, keine Commit-Behandlung, etc.
  - oft weitgehende Serialisierung globaler Transaktionen

OMG Object Management Architecture (OMA)

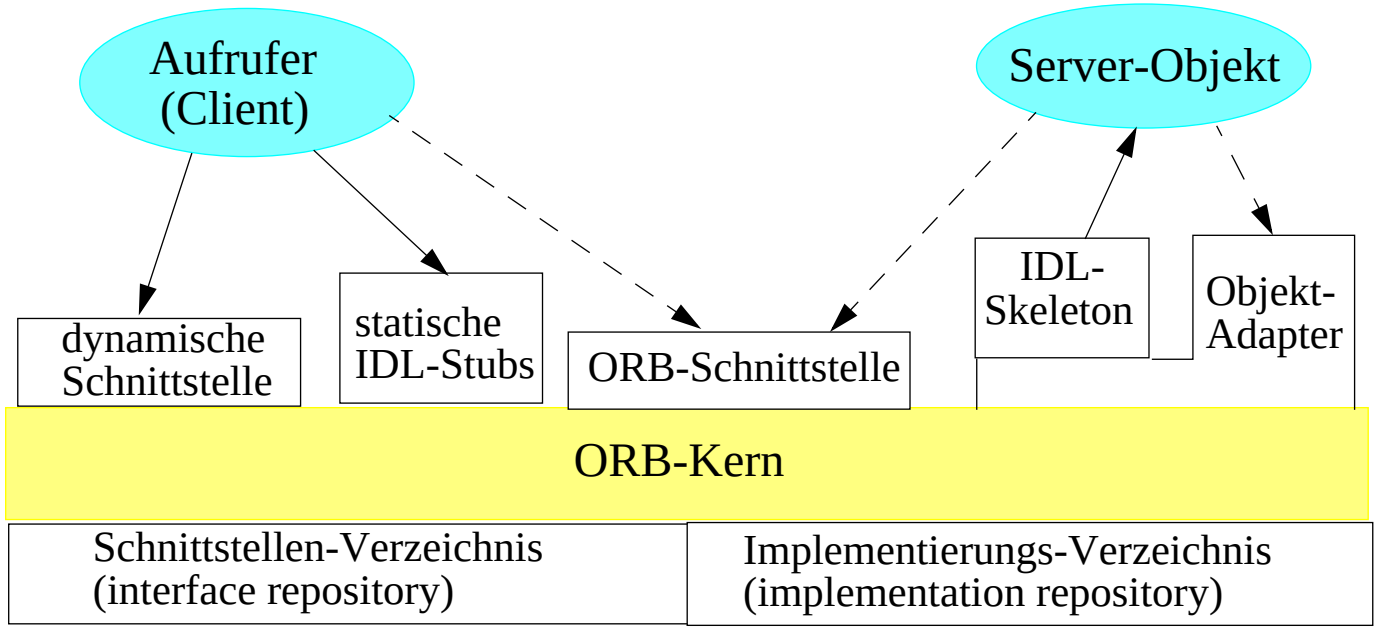


- Interoperabilität durch standardisierte Zugriffsschnittstellen
- Schnittstellenbeschreibung durch IDL (Interface Definition Language)
- Object Request Broker: Vermittler zur Client-/Server-Kooperation zwischen Objekten
  - Aufruf besteht aus: Operationsname, Zielobjekt, Parameter
  - Ortstransparenz
- Object Services: Basisfunktionen zur Realisierung objekt-orientierter Systeme (Erzeugung/Verwaltung von Klassen und Objekten, Namensverwaltung, Persistenz)
- Common Facilities: allgemeine Hilfsfunktionen (Klassenbibliotheken)

OMA (2)


Anwendungsobjekte
verfügbare Einrichtungen
Auftrags-annahme
Rechnungs-stellung
Reparatur
• • •
unternehmensweite Dienste
Object Request Broker
DBMS
Zeit
Namensdienst
Sicherheit
Registrierung
. . .
Anfrage
Transaktion
Regeln
. . .
Fenster-Manager
Formular-Manager
Desktop-Manager
Konfigurations-Manager
. . .
Objekt-Dienste

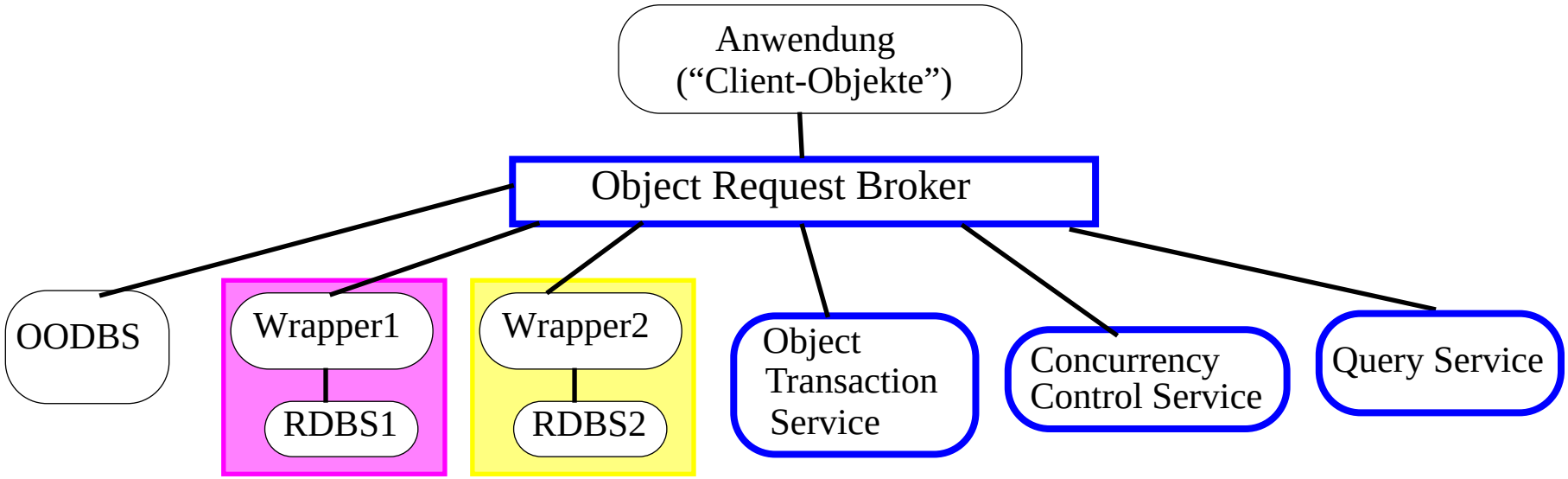
Common Object Request Broker Architecture (CORBA)



- statische und dynamische Methodenaktivierungen (Aufrufschnittstellen)
- ORB-Schnittstelle: Zugriff auf Infrastrukturfunktionen
  - Verwaltung globaler OIDs, Registrierung von Objekten, ...
- Kommunikation zwischen ORBs über IIOP (Internet Inter-ORB-Protokoll)

CORBA (2)

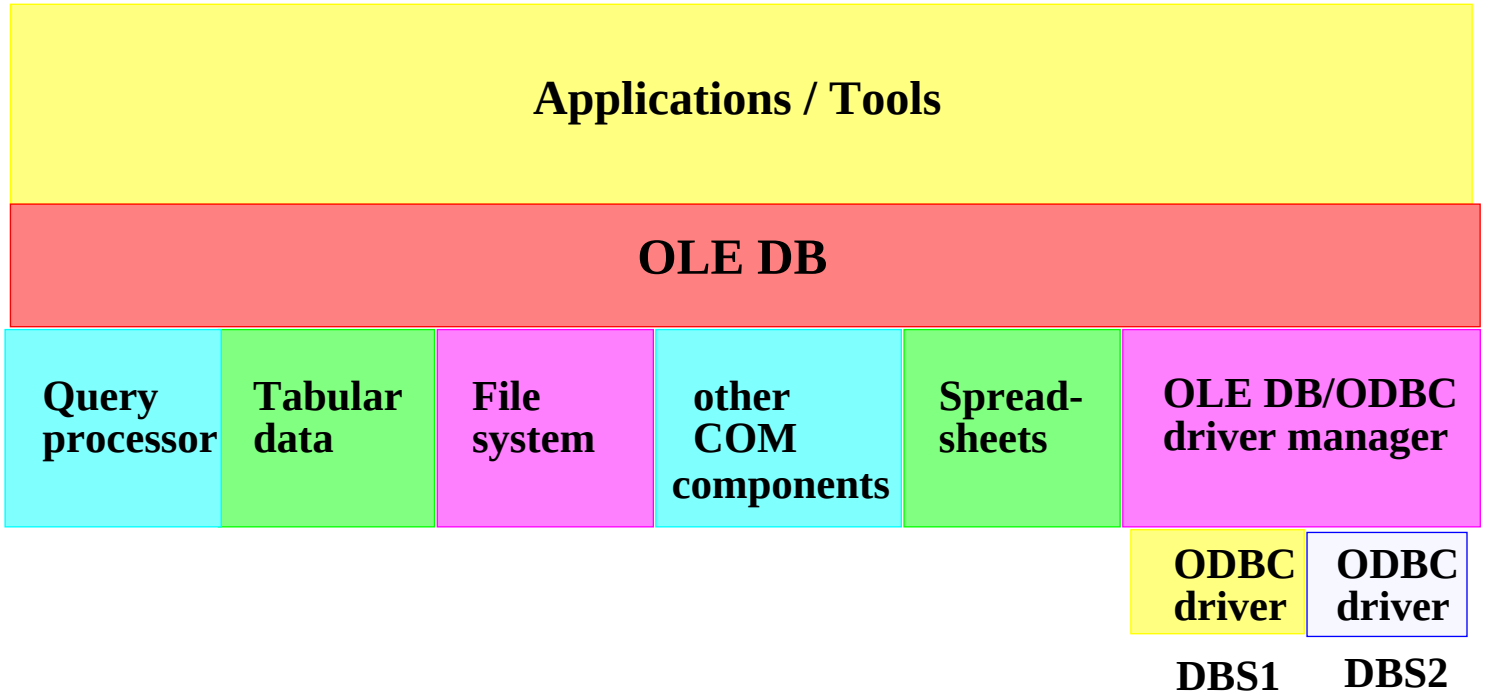
- standardisierter Aufruf von Methoden (Anwendungsprogrammen) über Object Request Broker



- Zugriff auf nicht-objektorientierte DBS kann über bereitzustellende Anwendungsprogramme (Server-Objekte) bzw. Wrapper erfolgen
- Object Transaction Service (OTS):
  Unterstützung für verteilte Transaktionen (2PC), geschachtelte Transaktionen
- Concurrency Control Service (CCS): global nutzbare Synchronisationsdienste
- Query Service: Anfragen auf OODBS und SQL-DBS

Microsoft OLE DB*

- Zugriff auf heterogene Datenquellen über festgelegte COM-Schnittstellen
  - OLE: Object Linking and Embedding; COM = Common Object Model (=> ActiveX)
- Beteiligte Komponenten:
  - Data Providers: SQL-DBS (Zugriff über ODBC), Dateien, Spreadsheets, E-Mail-Archive, ...
  - Data Consumers: Anwendungen, Tools
  - Data Service Providers: Query-Systeme u.ä.

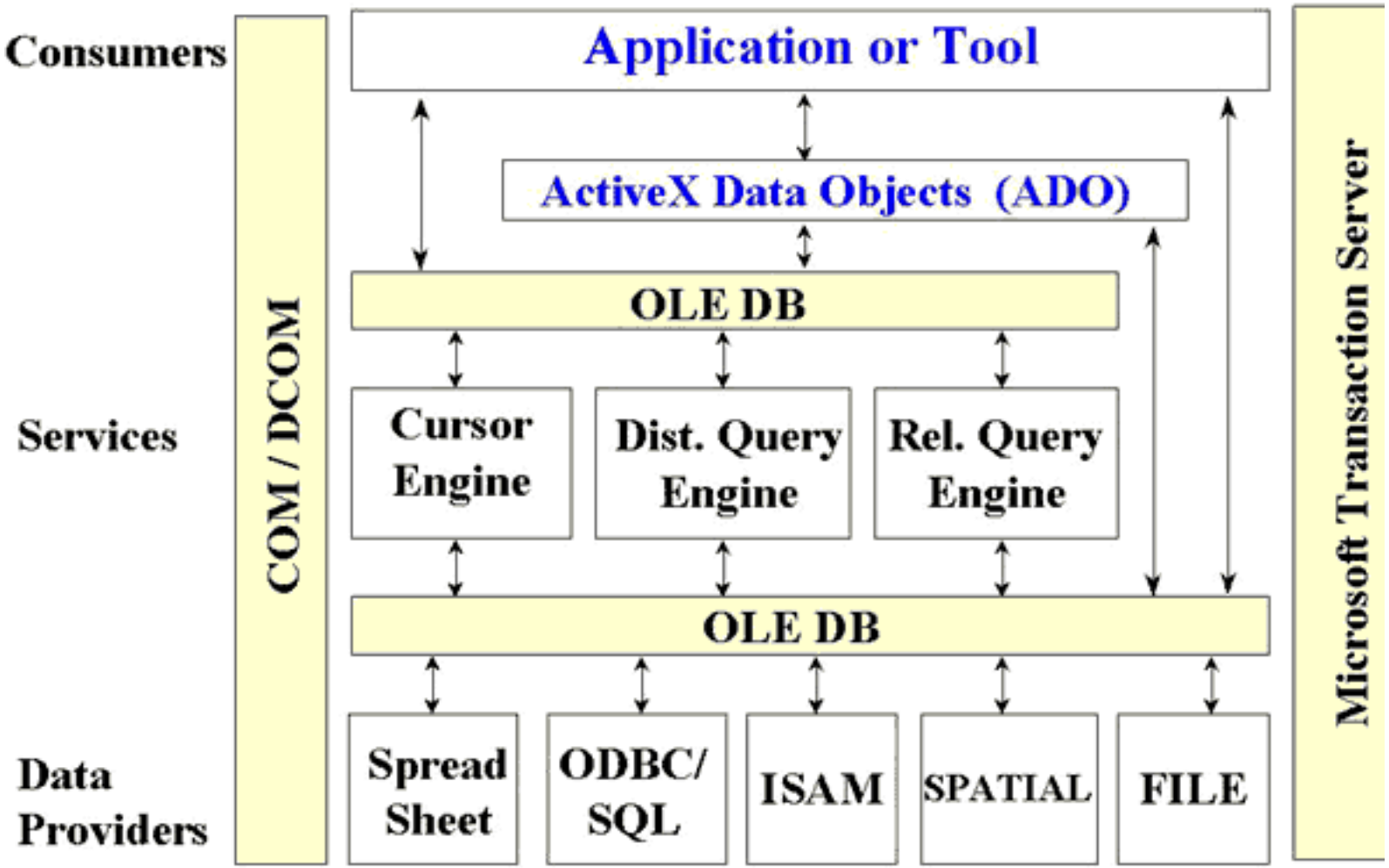


* http://www.microsoft.com/data

Microsoft OLE DB (2)

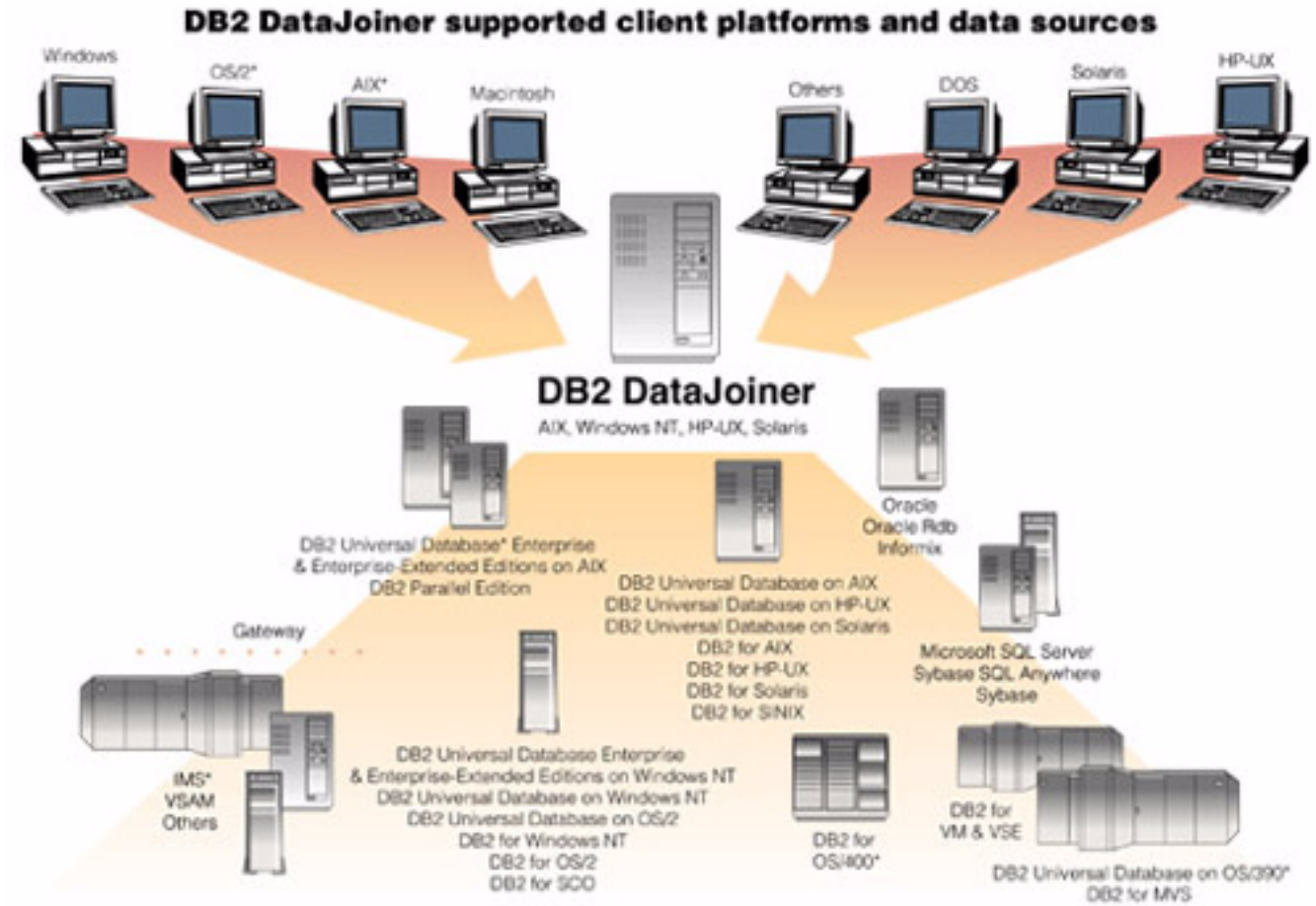
- Voraussetzung: tabellarische Daten
- Verwendung von 7 Objekttypen (mit insgesamt 55 Schnittstellen):
  - DataSources
  - DBSessions
  - Commands
  - Rowsets
  - Indexes
  - Errors
  - Transactions
- unterstützt Interoperabilität zwischen mehreren Datenquellen (verteilte Queries)
- verteilte Transaktionen sollen über Microsofts TP-Monitor unterstützt werden
- Unterschiede gegenüber ODBC
  - komponenten-basierte Architektur
  - COM-API statt C-API
  - Berücksichtigung von Nicht-SQL-Systemen
  - verteilte Queries/Transaktionen geplant

Microsoft Universal Data Access



IBM DB2 Data Joiner*

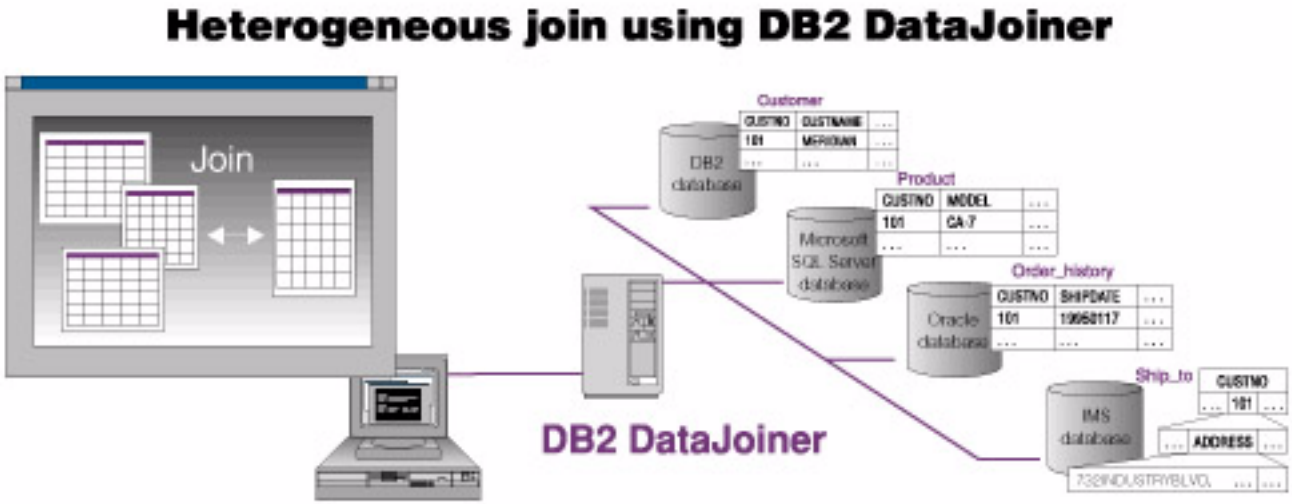
- föderative DBS-Lösung für einheitlichen Zugriff (DB2 SQL) auf zahlreiche SQL-DBS
- Unterstützung zahlreicher Client- und Server-Plattformen



* http://www.software.ibm.com/data/datajoiner/

IBM DB2 Data Joiner (2)

- hohe Funktionalität
  - verteilte/globale Anfragen (Joins) mit umfassender Query-Optimierung
  - Unterstützung von objekt-relationalen Eigenschaften (UDTs, UDFs, LOBs, rekursive Queries)
  - verteilte Transaktionen (XA-Unterstützung)
  - replizierte Datenhaltung



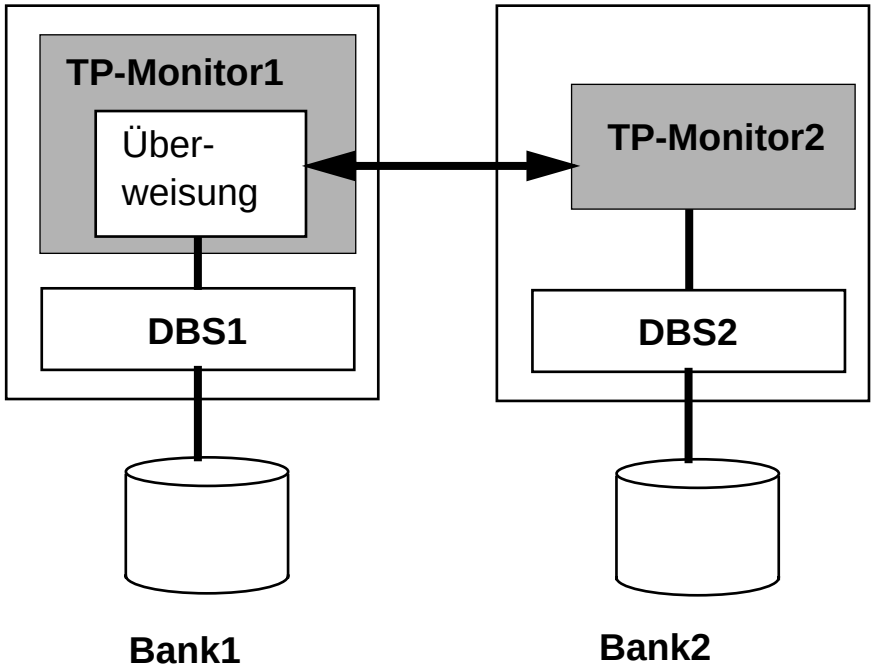
- Ortstransparenz über Nicknames

```
CREATE NICKNAME D_KUNDE FOR DB2.J15USER3.CUSTOMER
CREATE NICKNAME O_LIEFERUNGEN FOR ORACLE..J15USER3.ORDER_HISTORY

        SELECT D.CUSTNAME
        FROM D_KUNDE D, O_LIEFERUNGEN O
        WHERE  D.CUSTNO = O.CUSTNO AND O.SHIPDATE = 19990401
```

Zusammenfassung

- Heterogenität sowie Knotenautonomie erschweren
  - Verteilungstransparenz
  - einheitliche DB-Schnittstelle
  - Wahrung der Transaktionseigenschaften
- Einsatz von Standards notwendig
  - Kommunikation (Bsp. ISO RDA, DRDA), Transaktionsverwaltung (v.a. Commit-Protokoll) , APIs (SQL)
  - Alternative: Gateways
- Programmierte Verteilung (Verteilte Transaktionssysteme)
  - unterstützt Heterogenität und hohe Autonomie der LDBS
  - eingeschränkte Verteilungstransparenz
  - relativ einfache Realisierbarkeit
- Verteilung von DB-Operationen
  - gemeinsame Anfragesprache (gemeinsame SQL-Version) wünschenswert
  - Programmierer sieht mehrere Schemata
- Föderative DBS: gemeinsame DB-Sprache, Behandlung semantischer Heterogenität
- Integration von Nicht-DB-Daten zunehmend wichtig

Verteilung von DB-Operationen: Beispiel



Transaktionsprogramm enthält DB-Operationen für lokale und entfernte DBS

Eingabenachricht (Parameter) lesen

BEGIN WORK

CONNECT TO R1.DB1 ...

```
UPDATE  ACCOUNT
SET  BALANCE = BALANCE - :S
WHERE  ACCT_NO = :K1;
```

CONNECT TO R2.DB2 ...

```
UPDATE  GIROKTO
SET  KSTAND = KSTAND+ :S
WHERE  KNUMMER = :K2;
```

• • •

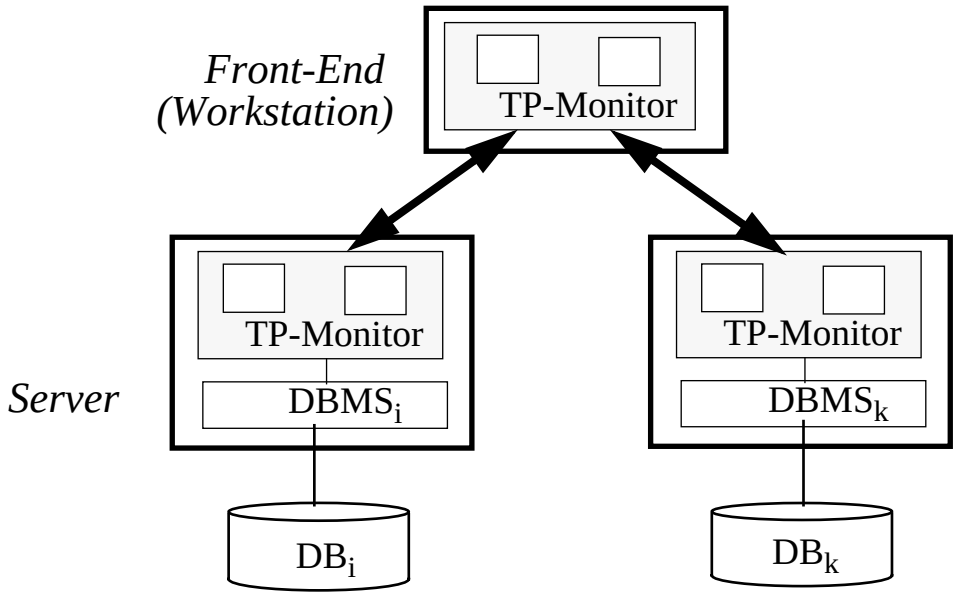
COMMIT WORK

DISCONNECT ...

Ausgabenachricht ausgeben

Verteilte Transaktionsverarbeitung in Client/Server-Systemen

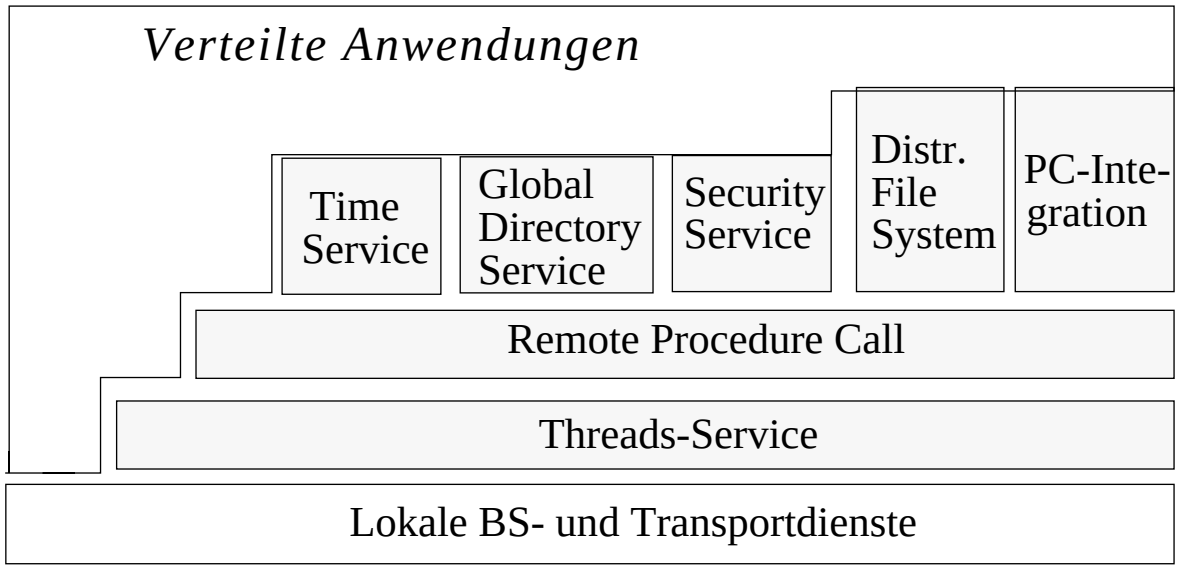
- Front-End (FE)- vs. Back-End-Verarbeitung
- Generell sinnvoll: Benutzerschnittstelle (GUI) sowie Nachrichtenverarbeitung im Front-End ("intelligentes" Terminal, PC, Workstation)
- Programmierte Verteilung: TAP auf FE-Rechner, Aufruf von Teilprogrammen auf Server



- TAP auf FE-Rechner,Ausführung von DB-Operationen auf Server
- TAP auf FE-Rechner (WS); DB-Verarbeitung auf FE und Server verteilt
  => Non-Standard-DBS, objektorientierte DBS
- keine Commit-Koordinierung im Front-End !

Distributed Computing Environment (DCE)

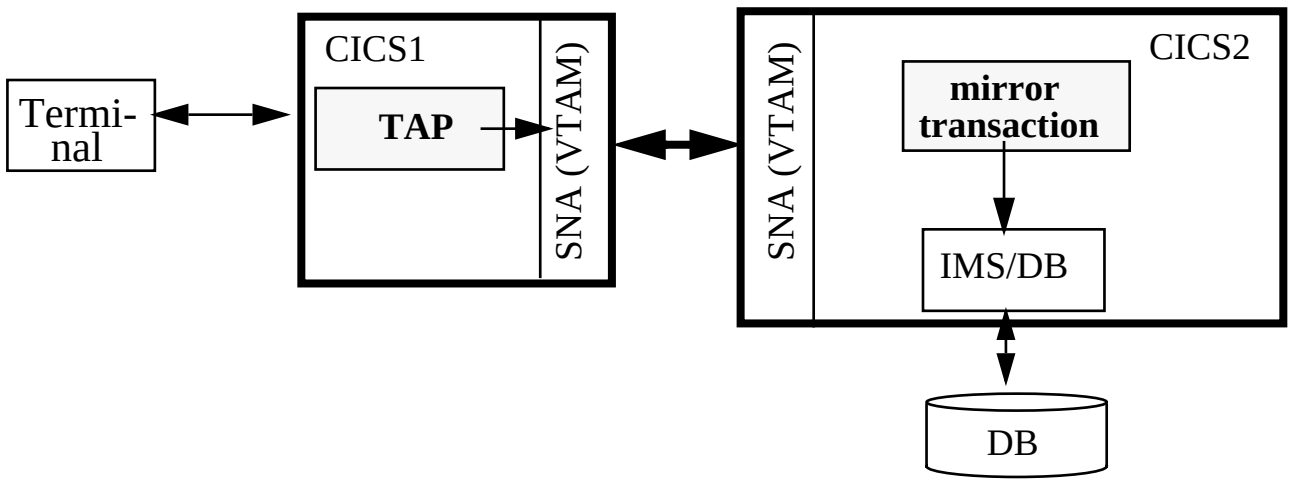
- entwickelt vom Herstellerkonsortium OSF (Open Software Foundation, derzeit 74 Firmen)
  - Threads, Remote Procedure Call (RPC)
  - Naming-Service (X.500-Directories), Schutzmechanismen (Kerberos)
  - verteilte Dateiverwaltung (Andrew File System, AFS), globale Uhrensynchronisierung, etc.



- Realisierungen für zahlreiche BS: OSF/1, DEC/Ultrix, IBM/AIX, SunOS, VAX/VMS, ...
- keine Transaktionsverwaltung
- Realisierung von “Middleware” basierend auf DCE:
  - TP-Monitore (Bsp.: Transarc Encina, IBM CICS/6000, DEC ACMS)

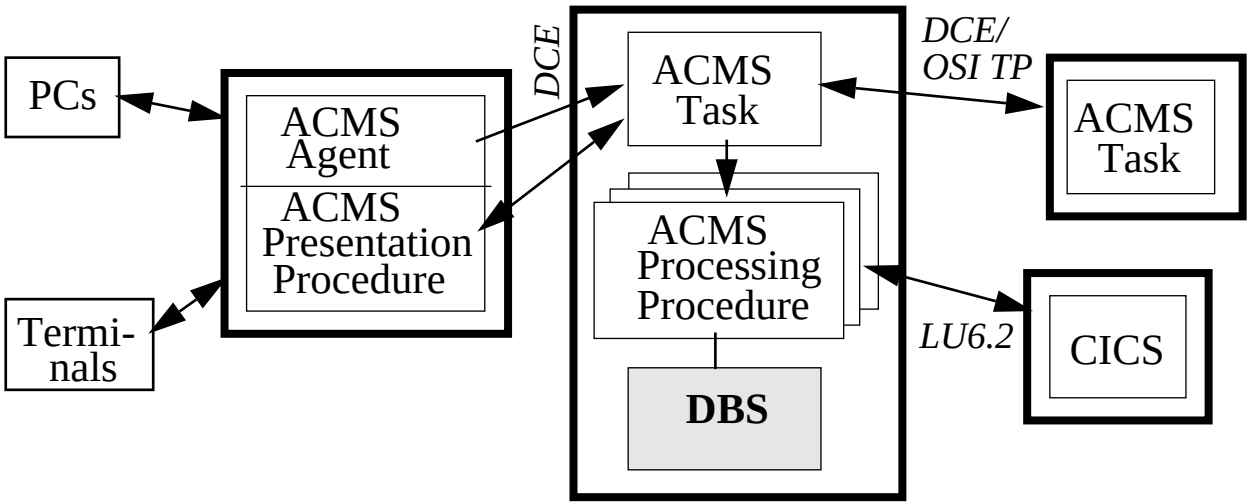
CICS Function Request Shipping

- Verschicken einzelner DL/1-Befehle bzw. einfacher Dateizugriffe im Rahmen einer eigenen VTAM-Session
- CICS gewährleistet Ortstransparenz
- Starten einer “mirror transaction” im entfernten CICS-System
- verteilte Commit-Behandlung nur für ändernde Teil-Transaktionen (für Lese-Befehle werden entfernte Teil-Transaktion sowie Session direkt terminiert)
- sehr hoher Overhead



DEC ACMS

- modulare Trennung von Client- und Server-spezifischen Anwendungsteilen
  - Client-Teil: ACMS-Agent + Präsentationsdienste
  - Server-Teil: ACMS-Tasks + Verarbeitungsprozeduren



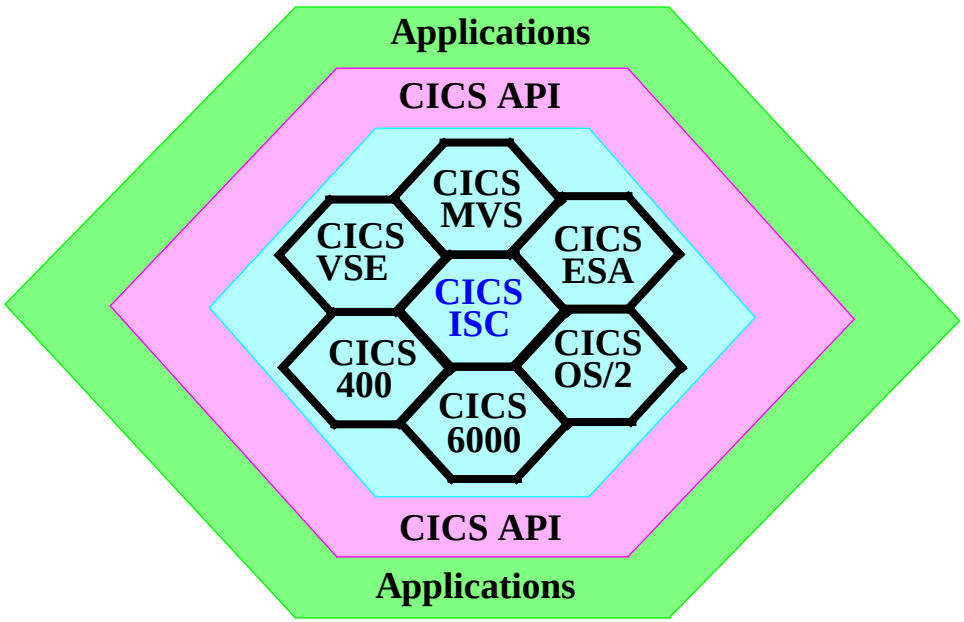
- Tasks
  - Spezifikation der Ablaufkontrolle, Transaktionsgrenzen und Ausnahmebehandlung
  - eigene *Task Definition Language* (TDL)

Beispiel: *START-TRANS;*
CALL Withdraw ();
CALL Deposit ();
END-TRANS;

- Kommunikation über transaktionsgeschützte RPCs
- Kooperation mit anderen TP-Monitoren (CICS, IMS/DC)
- Zugriff auf DB2-Datenbanken (Gateway: RdbAccess)
- (demnächst) Unterstützung folgender Standards: DCE, X/OPEN DTP, MIA STDL

CICS ISC (Intersystem Communication)

- CICS
  - bedeutendster TP-Monitor für Mainframes (> 10000 Installationen unter MVS)
  - Zusammenarbeit mit zahlreichen DBS
  - eingeführt 1968; Unterstützung verteilter Transaktionen bereits seit 1978
  - Portierung von CICS auf zahlreiche Plattformen von IBM sowie anderen Herstellern
- Kommunikation zwischen CICS-Instanzen: ISC
  - Transaction Routing
  - Function Request Shipping (= Verteilung von DL/1-Op.)
  - Distributed Transaction Processing (= Programmierte Verteilung)
  - asynchrone Verarbeitung (ohne Transaktionsschutz)



CICS Distributed Transaction Processing

- synchroner Prozeduraufruf in entferntem CICS-System
- kein RPC, sondern Peer-to-Peer-Kommunikation im Rahmen von Sessions (conversations)
- mehrere Interaktionen pro Session möglich
- geschachtelte Programmaufrufe möglich
- keine Ortstransparenz, da Kommunikationsbefehle im TAP
- Transaktionsverwaltung basiert auf SNA LU 6.2 aka APPC
  - LU = Logical Unit (logischer Kommunikationspartner, d.h. Programm)
  - LU6.2: transaktionsgeschützte Sessions
  - APPC: Advanced Program-to-Program Communication

Beispiel

Rechner R1

```
EXEC CICS
   ALLOCATE DebitCredit@R2.CICS2
   DATA ... USERID ..., PASSWORD ...,
   RETURNING sessionid;

EXEC CICS RECEIVE DATA
   SESSION sessionid
   DATA ... STATUS ...;

EXEC CICS SYNCPT;
```

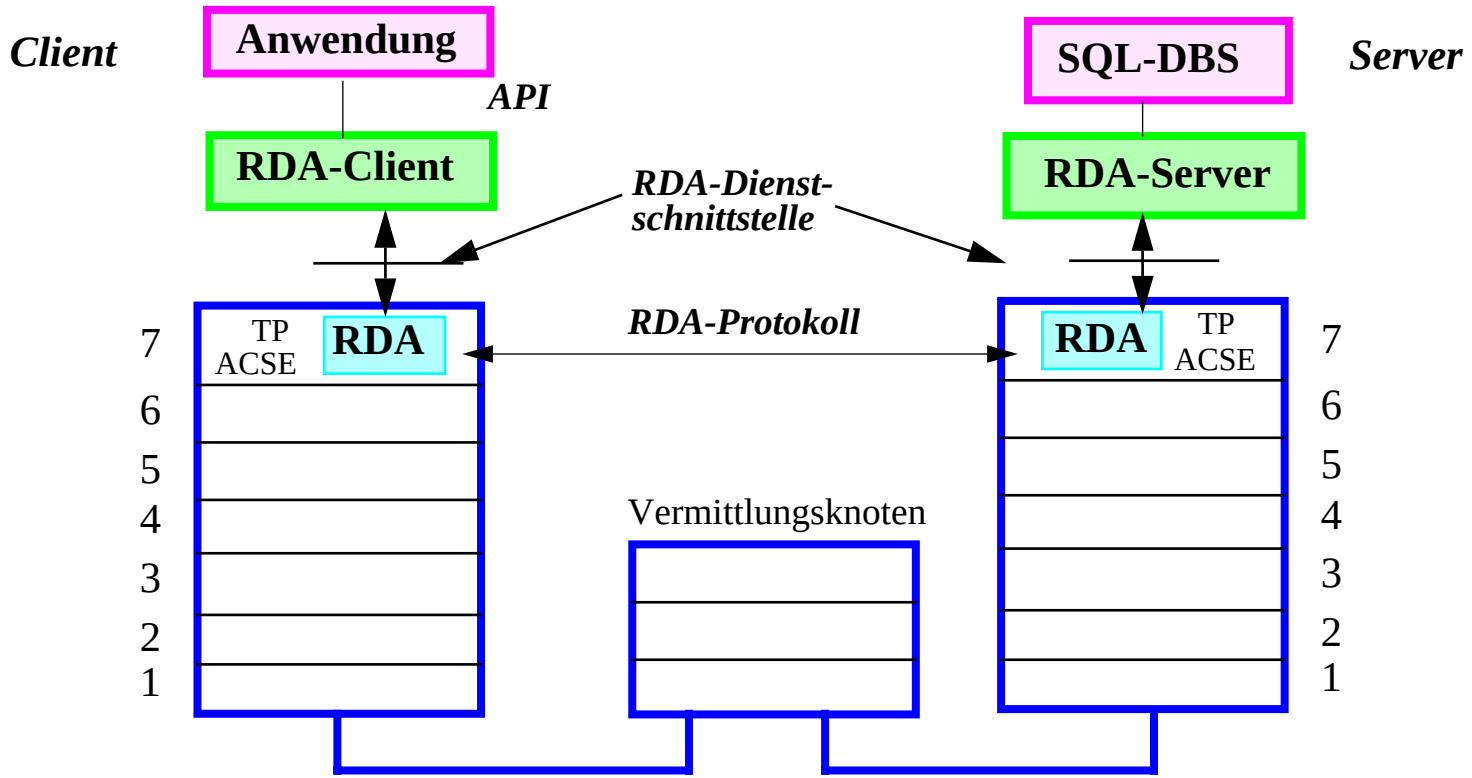
Rechner R2

```
do_debit-credit (input message);

EXEC CICS
   SEND DATA
   SESSION sessionid DATA ...;

EXEC CICS RETURN COMMIT status;
```

- Kooperation auch mit anderen TP-Monitoren, die LU6.2-Protokoll einhalten, möglich

Remote Database Access (RDA)

- RDA-Standard ermöglicht Fernzugriff auf Datenbanken in “offenen” Netzen
  - standardisierte Kommunikationsmechanismen zwischen DB-Anwendungen (Clients) und DB-Servern
- RDA ist Teil der Anwendungsebene (Schicht 7) des ISO OSI-7-Schichtenmodells



RDA (2)

- RDA-Spezifikation besteht aus zwei Teilen:
  - Generischer Teil
  - Spezialisierungen (derzeit nur für SQL)

- Bisherige Entwicklungsgeschichte

  1985: Beginn der RDA-Arbeiten im Rahmen der ECMA

  1987: Beginn der ISO-Standardisierung

  1993: Standards für Generic RDA (ISO/IEC 9579-1) sowie
  RDA SQL-Spezialisierung (ISO/IEC 9579-2) für SQL-89 sowie Entry-SQL-92

- Wesentliche Merkmale
  - verbindungsorientierte Kommunikation
    => Anwendung muß Verbindung mit Servern explizit eröffnen/beenden
  - Aufrufeinheiten: SQL-Anweisungen
  - Single-Server- oder Multi-Server-Variante

- Nutzung anderer OSI-Dienste
  - Einrichtung von Netzwerkverbindungen durch ACSE (Association Control Service Element)
  - verteiltes Commit-Protokoll über TP bzw. CCR (Commitment, Concurrency and Recovery)

RDA (3)

- RDA-Kommunikationsdienste:

Dialogverwaltung:
- R- Initialize
- R-Terminate

Ressourcen-Verwaltung:
- R-Open
- R-Close

Transfer von DB-Operationen:
- R-ExecuteDBL
- R-DefineDBL
- R-InvokeDBL
- R-DropDBL

Kontrolldienste:
- R-Status
- R-Cancel

Transaktionsverwaltung:
- R-BeginTransaction
- R-Commit
- R-Rollback

- RDA-Kommunikationsprotokoll
  - spezifiziert genaue RDA-Nachrichtenformate
  - Reihenfolgeregeln (z.B. R-Initialize muß erste Anweisung sein)

- RDA definiert kein API (unterstellt Verwendung von Standard-SQL)

SQL Access

- Ziele:
  - Fernzugriff auf SQL-Server auf RDA-Basis (Interoperabilität zwischen relationalen DBS)
  - Festlegung von API und FAP (Formats and Protocols)
  - Prototyp-Implementierung
- Realisierung durch Herstellerkonsortium *SQL Access Group* (gegründet 1989)
- API
  - gemeinsamer SQL-Dialekt
  - normierte Fehlercodes und Schemaangaben
  - Session Management (CONNECT, DISCONNECT)
  - direkte Einbettung und CLI (Call Level Interface)
- FAP entspricht RDA-Protokoll
- Beispiel:

Anwendung	***erzeugte RDA-Aufrufe***
CONNECT TO "SYBASE@a.b.c.d" AS "Sybase" USER "E_Rahm"	*A-Assoc <...>* *R-Initialize < "E_Rahm",...>* *R-Open <"SYBASE@a.b.c.d", ...>*
INSERT INTO Pers VALUES (...)	*R_BeginTransaction <...>* *R_ExecuteDBL <"INSERT",...>*
COMMIT WORK	*R_Commit <...>*
DISCONNECT "Sybase"	*R_Close <"Sybase",...>* *R_Terminate <...>*

IBMs Distributed Relational Database Architecture (DRDA)

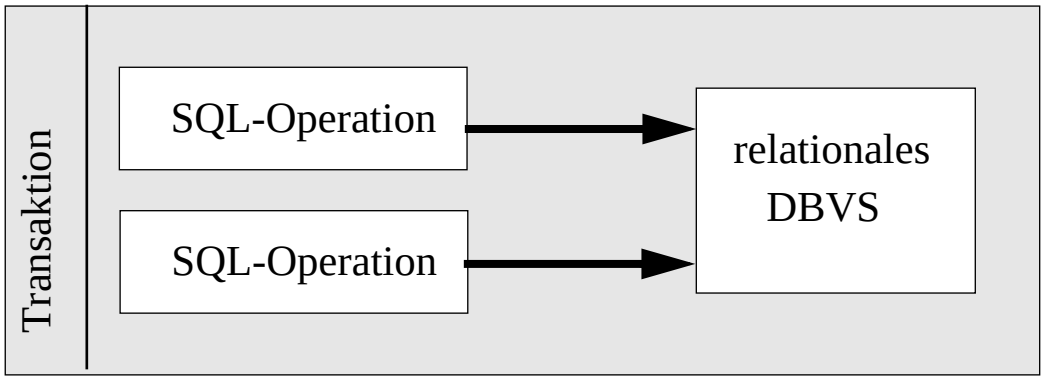
- Ziel: Interoperabilität zwischen SQL-DBS von IBM
  - DB2 (MVS-Betriebssystem)
  - DB2/6000 (AIX, RS/6000)
  - DB2/2 (OS/2, PS/2)
  - SQL/DS (VM)
  - SQL/400 (OS/400, AS/400)
  - DRDA-kompatible Fremd-DBS

- Unterschiedliche Kooperationsformen
  - Remote Request
  - Remote Unit of Work
  - Distributed Unit of Work
  - Distributed Requests

- Remote Request
  - Weiterleitung einzelner SQL-Anweisungen (Requests) an entferntes DBS
  - Transaktion = 1 DB-Operation
  - Einsatz v.a.für PC-basierte DB-Zugriffe)

DRDA (2)

- Remote Unit of Work



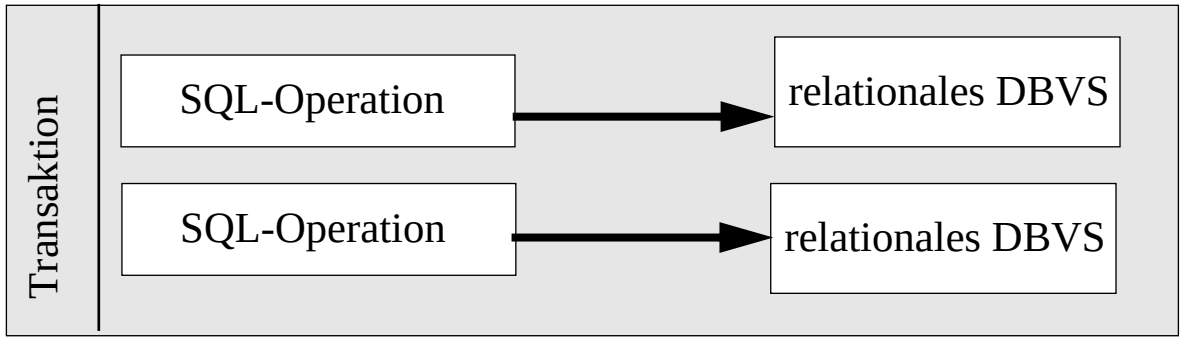
- Ausführung mehrerer entfernter SQL-Anweisungen
- alle SQL-Anweisungen einer Transaktion auf 1 DBS beschränkt
- Anwendung kontrolliert Verbindungen mit DBS (keine Verteilungstransparenz)
- Kommunikation pro SQL-Befehl

Beispiel

```
CONNECT TO L.DB2 ...
SELECT ... FROM PERS1 ...
COMMIT WORK;
CONNECT TO F.DB2 ...
SELECT ... FROM PERS2 ...
UPDATE PERS2 SET ...
COMMIT WORK;
DISCONNECT ...
```

DRDA (3)

- Distributed Unit of Work



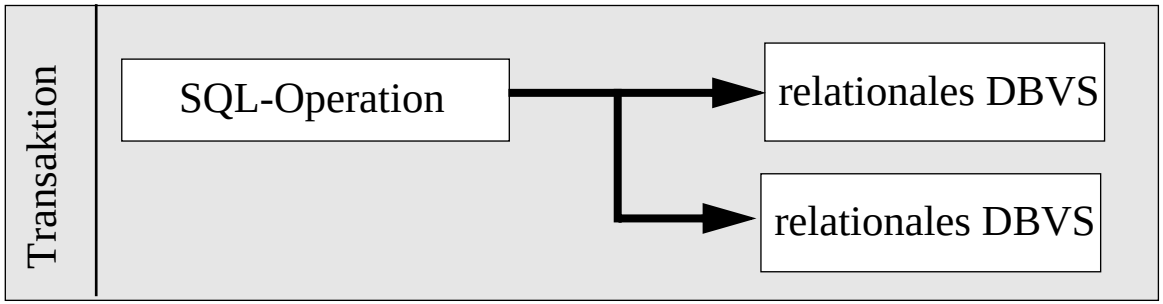
- mehrere SQL-DBS pro Transaktion möglich
- jede SQL-Anweisung auf 1 DBS beschränkt
- korrespondiert zu Verteilung ganzer DB-Operationen
- 4 Varianten: multi-site read, local-site update, single-site update, multi-site update
- z.Zt. noch kein *multi-site update*: Änderungen sowie Integritätsbedingungen (!) auf Daten eines Rechners beschränkt

Beispiel

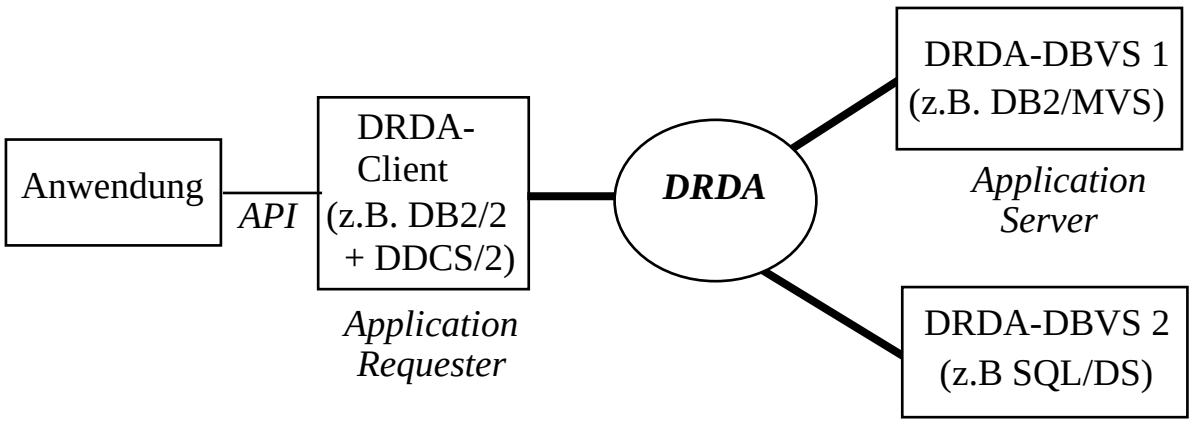
```
SELECT ... FROM PERS1 ... (= L.RAHM.PERS1)
SELECT ... FROM PERS2 ... (=F.ADMIN.PERS2)
UPDATE PERS2 SET ...
COMMIT WORK;
```

DRDA (4)

- Distributed Requests
  - verteilte SQL-Anweisungen
  - DB-Verteileinheit: ganze Relationen
  - z.Zt. noch nicht unterstützt



- Beispiel einer Ablaufumgebung



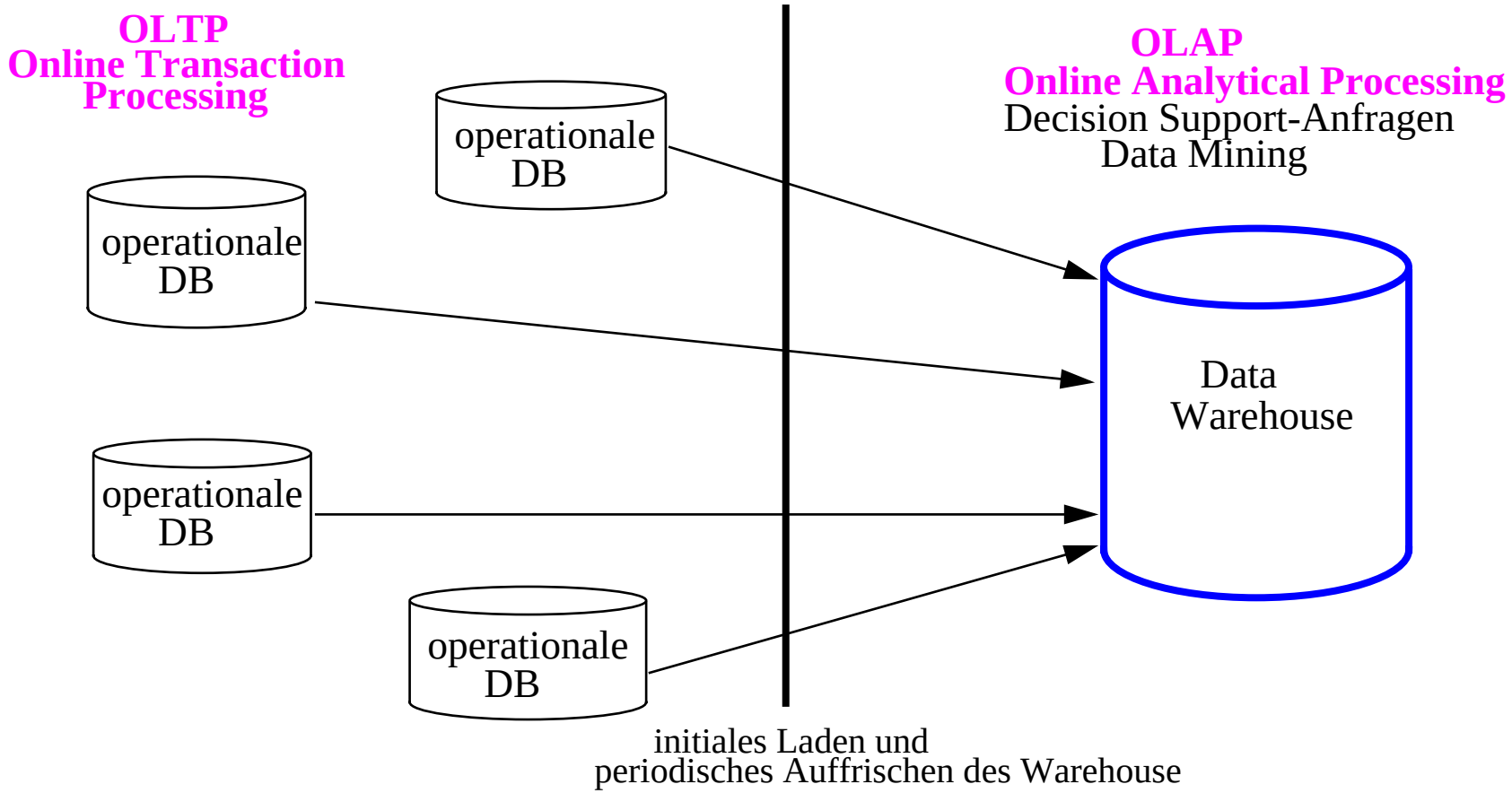
  - DRDA definiert Nachrichtenformate und -protokolle
  - Abbildung SQL <-> DRDA-Nachrichten über Application Requester/Server

8. Data Warehouses

- Einführung
- Fallbeispiele
- DW-Eigenschaften, Architektur
- Schritte beim Aufbau eines DW
- Datenmodellierung
  - Benutzersicht, Operationen (Drill down, roll up ...)
  - ROLAP vs. MOLAP
  - Star-Schema, Snowflake-Schema
- Data Mining

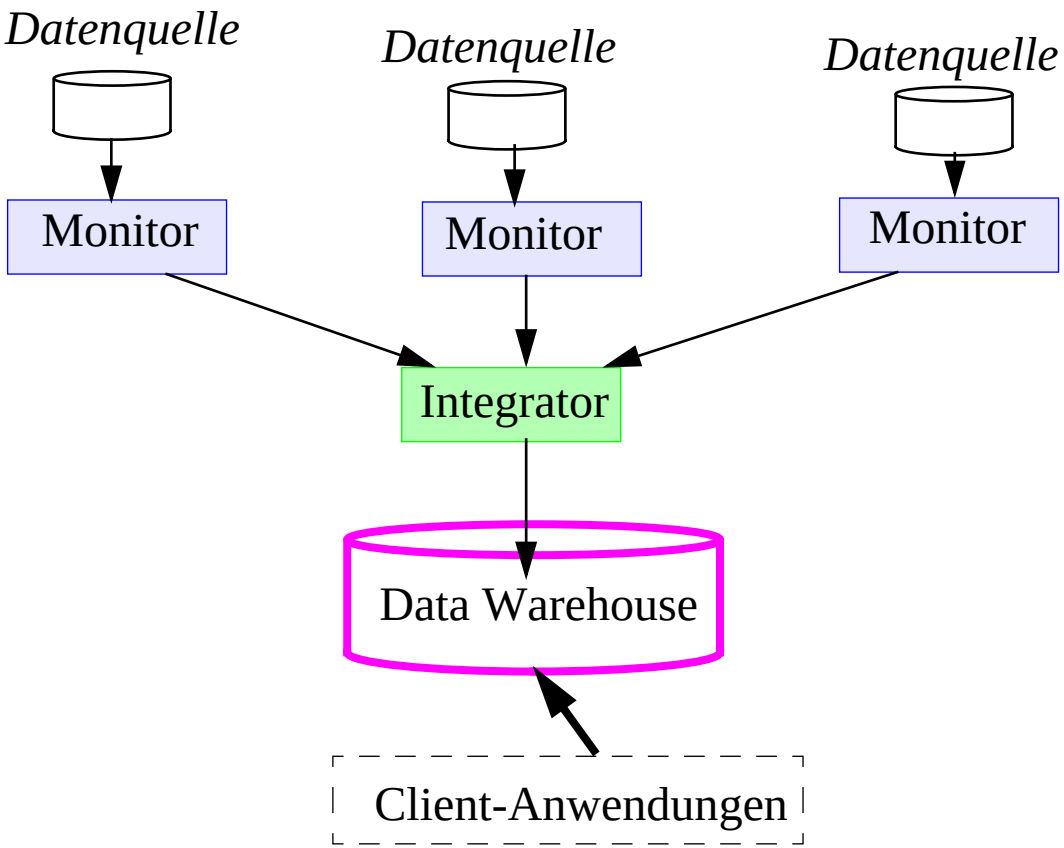
Data Warehouses

- Ausgangsproblem: viele Unternehmen haben Unmengen an Daten, ohne daraus ausreichend Informationen und Wissen für kritische Entscheidungsaufgaben ableiten zu können
- Zusammenführung (Integration) und Verdichtung (Aggregation) von Daten aus mehreren, i.a. heterogenen Quellen in zentraler Datenbank
  - Ziel: schnelle Reaktion auf sich ändernde Marktanforderungen ermöglichen



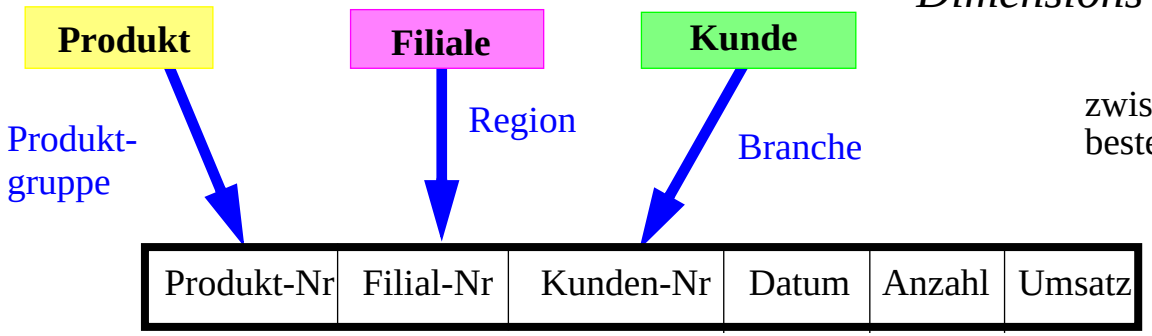
Data Warehouse (2)

- weitere Anforderungen
  - komplexe, mehrdimensionale Aggegierungsaufgaben
  - Berücksichtigung von Zeit
  - umfangreiche Anfragen u. große Datenmengen erfordern oft Einsatz eines Parallelen DBS



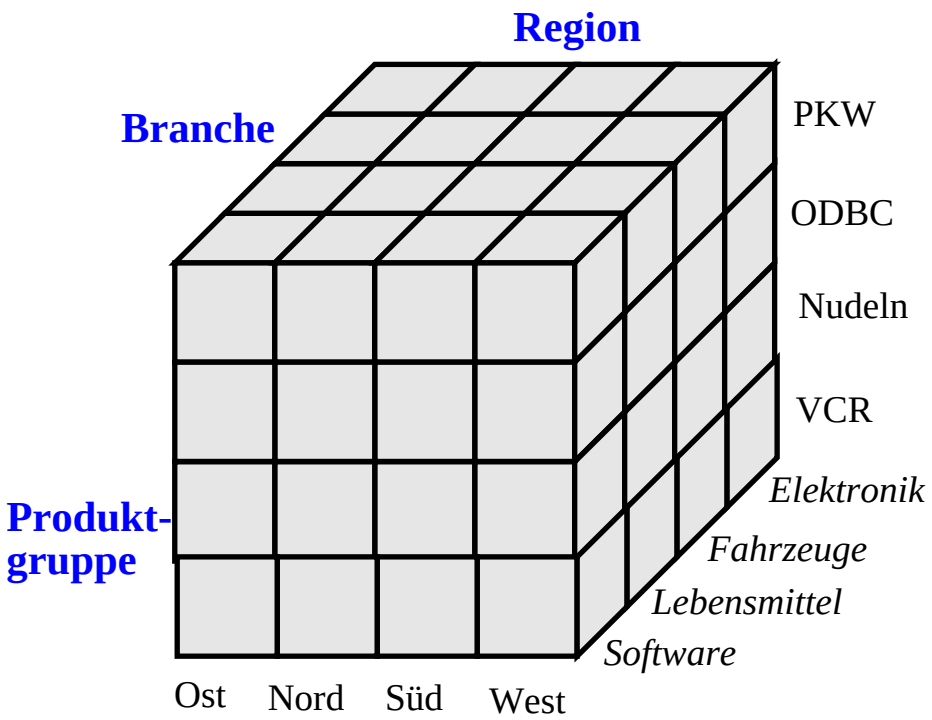
Data Warehouse - Beispiel

Dimensions-Relationen



zwischen Detail-Relation und Dimensions-Relationen bestehen i.a. mehrfache n:1-Beziehungen

Detail-Relation (Faktentabelle)



Einsatzbeispiele

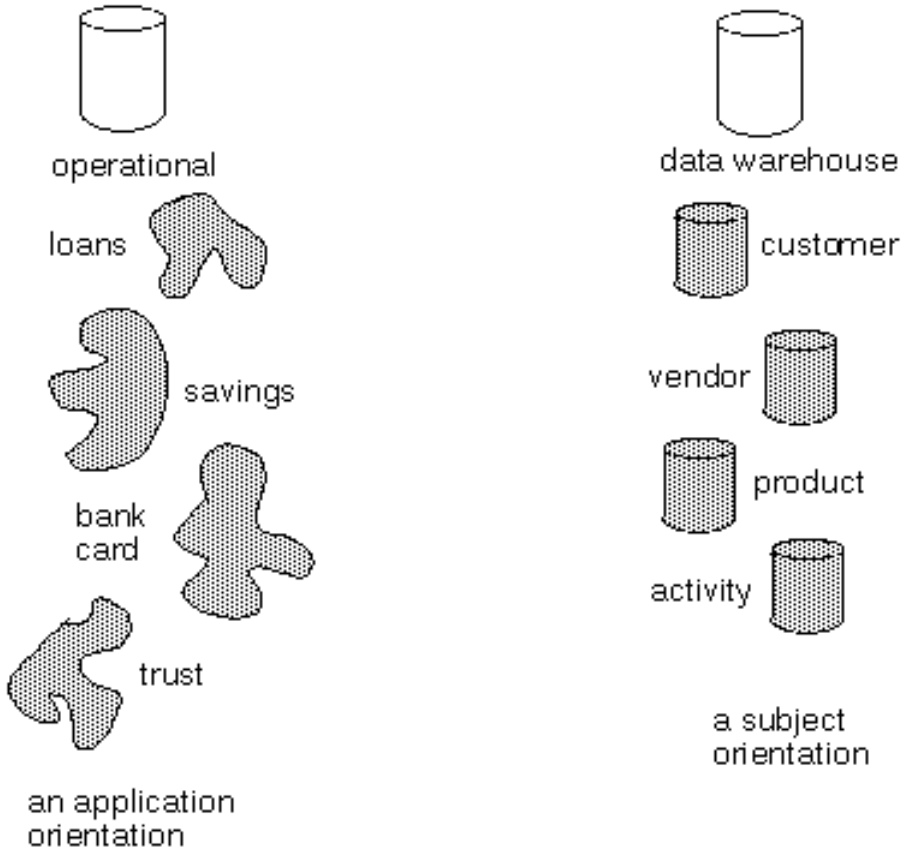
- Warenhauskette
  - Verkaufszahlen und Lagerbestände aller Warenhäuser
  - mehrdimensionale Analysen: Verkaufszahlen nach Produkten, Regionen, Warenhäusern
  - Analyse des Kaufverhaltens von Kunden
  - Erfolgskontrolle von Marketing-Aktivitäten
  - Minimierung von Beständen
  - Optimierung der Produktpalette
  - Optimierung der Preisgestaltung • • •

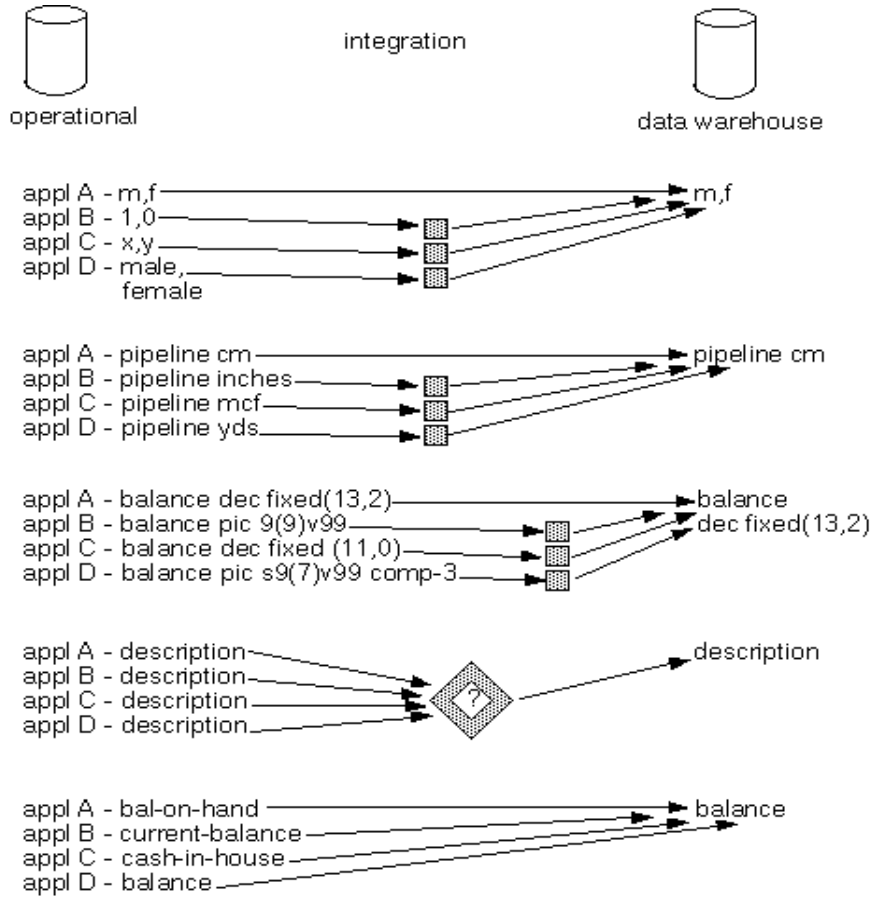
- Versicherungsunternehmen
  - automatische Risikoanalyse
  - schnellere Entscheidung über Annahme eines Kunden (Lebensversicherung; Krankenversicherung ...)
  - Bewertung von Filialen, Vertriebsbereichen, Schadensverlauf, ...

- Banken
- Versandhäuser
- Restaurant-Ketten • • •

Eigenschaften eines Data Warehouses

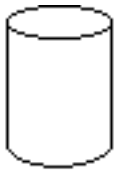
- Gemäß W. Inmon* ist ein Data Warehouse eine
  - subjekt-orientierte
  - integrierte
  - zeit-variante und
  - nicht-flüchtige (stabile) Kollektion von Daten zur Entscheidungsunterstützung





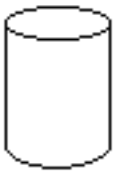
* W. Inmon: Building the Data Warehouse, 2nd ed., John Wiley 1996

Eigenschaften eines Data Warehouses (2)



operational

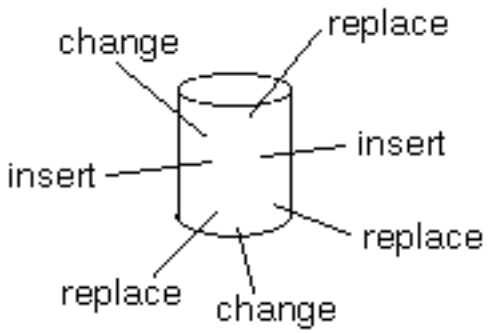
time variancy



data warehouse

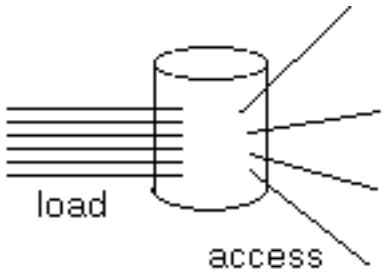
current value data:
- time horizon — 60-90 days
- key may or may not have an element of time
- data can be updated

snapshot data:
- time horizon — 5-10 years
- key contains an element of time
- once snapshot is made, record cannot be updated



operational

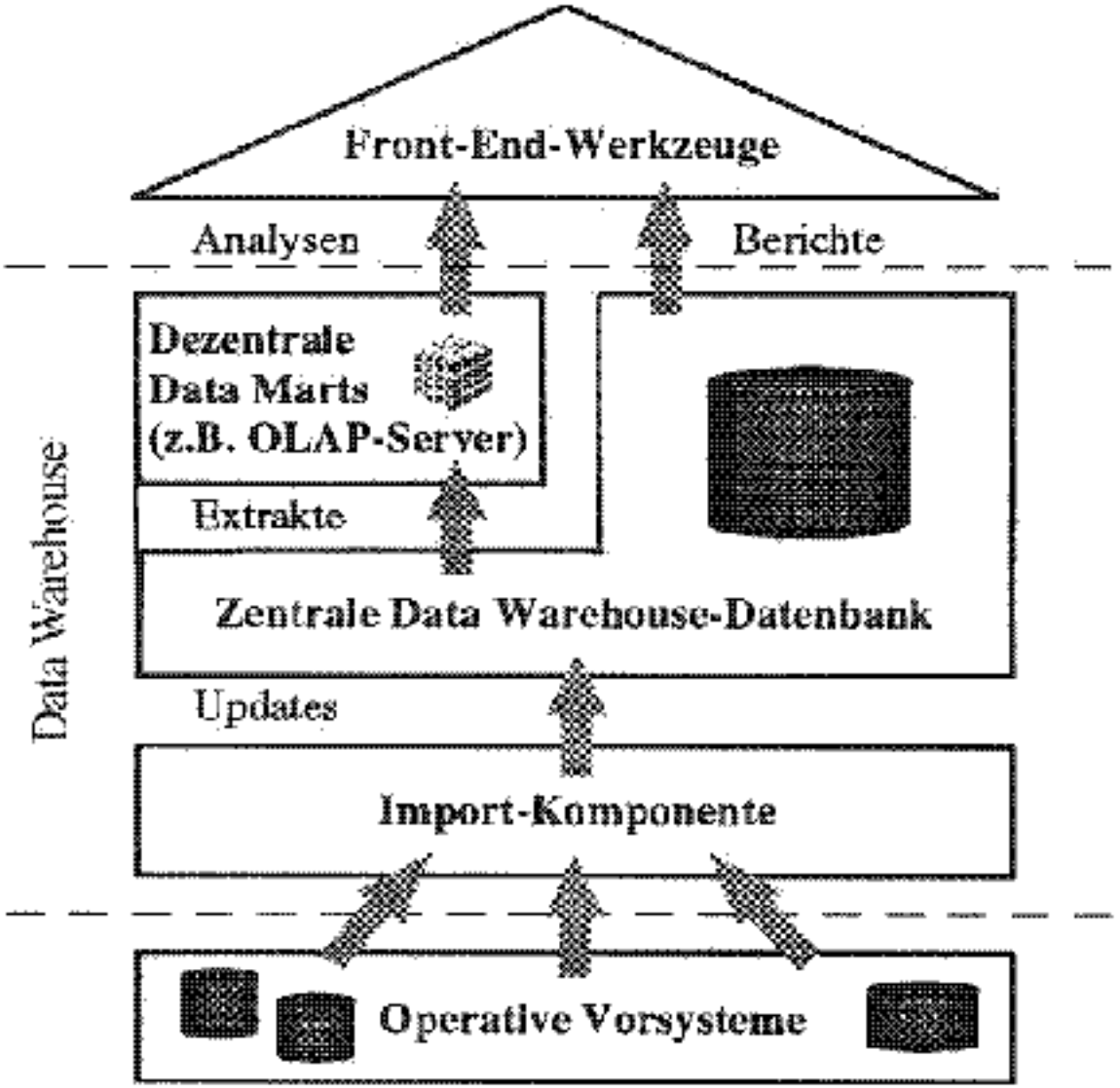
data is updated on a record-by-record basis regularly

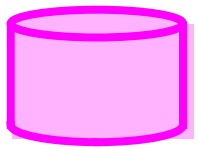


data warehouse

data is loaded into the warehouse and is accessed there, but once the snapshot of data is made, the data in the warehouse does not change

Grobarchitektur



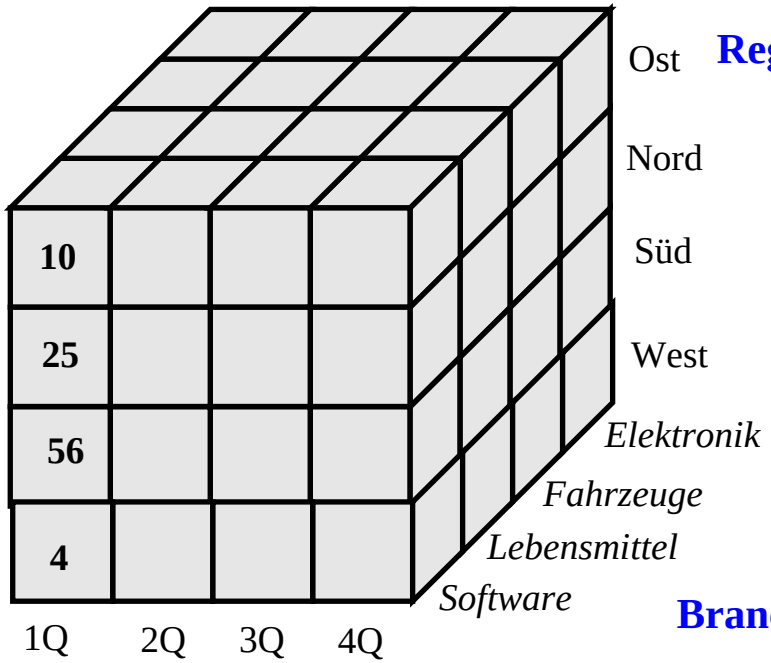


Metadaten-
Repository

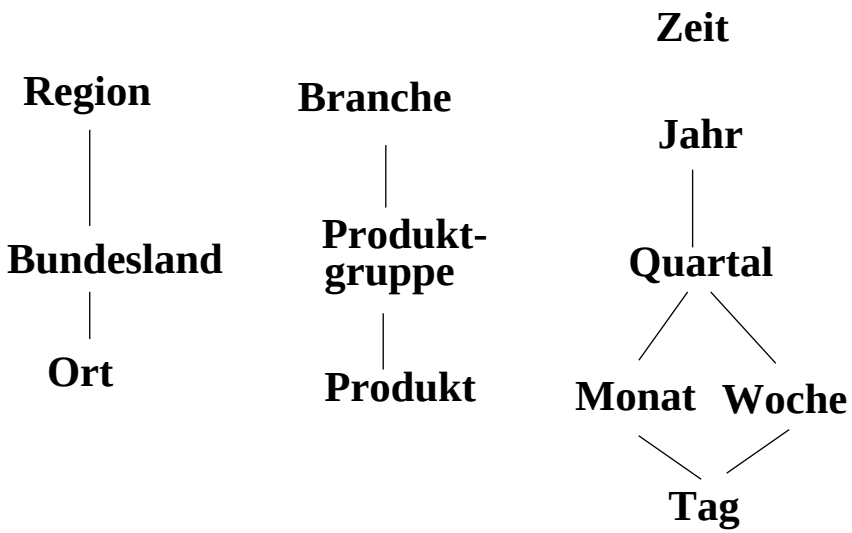
Schritte beim Einsatz eines DW

1. Definition der Architektur; Kapazitätsplanung; Auswahl der DBS, OLAP-Server und Tools
2. Integration der Server und Tools
3. Entwurf des Warehouse-Schema sowie der Views
4. Definition der physischen DW-Organisation, Datenallokation, Partitionierung, Zugriffspfade
5. Anbindung der Datenquellen über Gateways, ODBC-Treiber, Konverter/Wrapper
6. Entwurf und Implementierung von Skripten zur Datenextraktion, -bereinigung und -transformation sowie zum Laden und Refresh
7. Füllen des Metadaten-Repository mit Schema- und View-Definitionen, Skripten, etc.
8. Entwurf/Implementierung von Endbenutzer-Anwendungen
9. Inbetriebnahme von DW und Anwendungen

Mehrdimensionale Datensicht



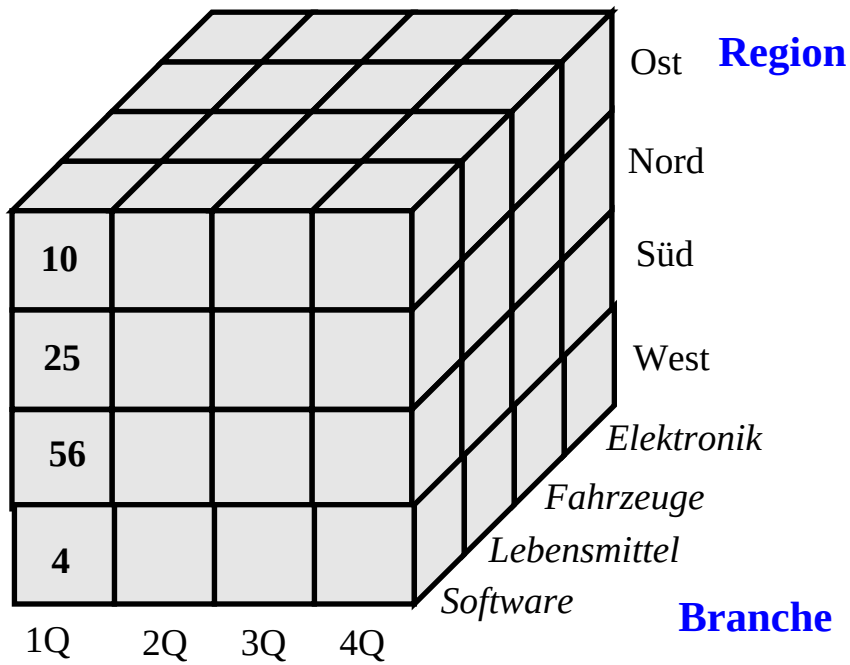
Hierarchische Dimensionierung



- Aggregierung (und Ranking) bezüglich unterschiedlicher Dimensionen
- weitere Operationen: Drill-Down, Roll-Up, Slice-and-Dice ...
- Unterstützung durch Front-End-Tools

Relational vs. multidimensional (ROLAP vs. MOLAP)

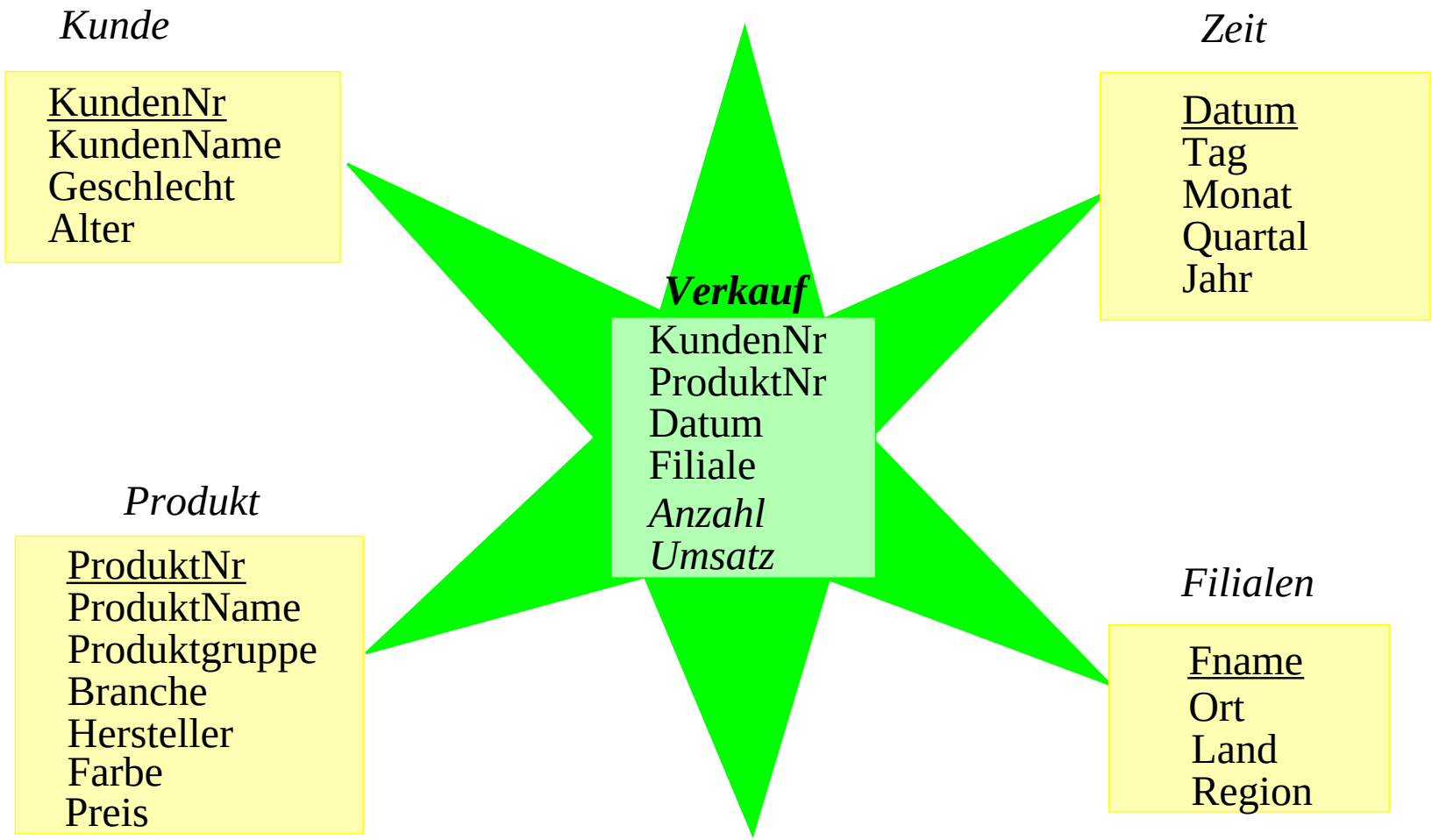
Bestellnr	Region	Branche	Zeit	Menge
1406	Ost	Fahrzeug	2Q	5
4123	West	Elektronik	1Q	58
7829	Süd	Fahrzeug	2Q	30
5327	Ost	Lebensmittel	4Q	3000
9306	Nord	Software	1Q	25
2574	Ost	Elektronik	4Q	2



- Relation: Untermenge des Kreuzproduktes aller Wertebereiche
  - nur vorkommende Wertekombinationen werden gespeichert (Tupel)
- multidimensionale Darstellung: Kreuzprodukt aller Wertebereich mit aggregiertem Wert pro Kombination
  - Annahme: fast alle Kombinationen kommen vor
  - Frage: auf welcher Ebene erfolgt die Aggregation ?

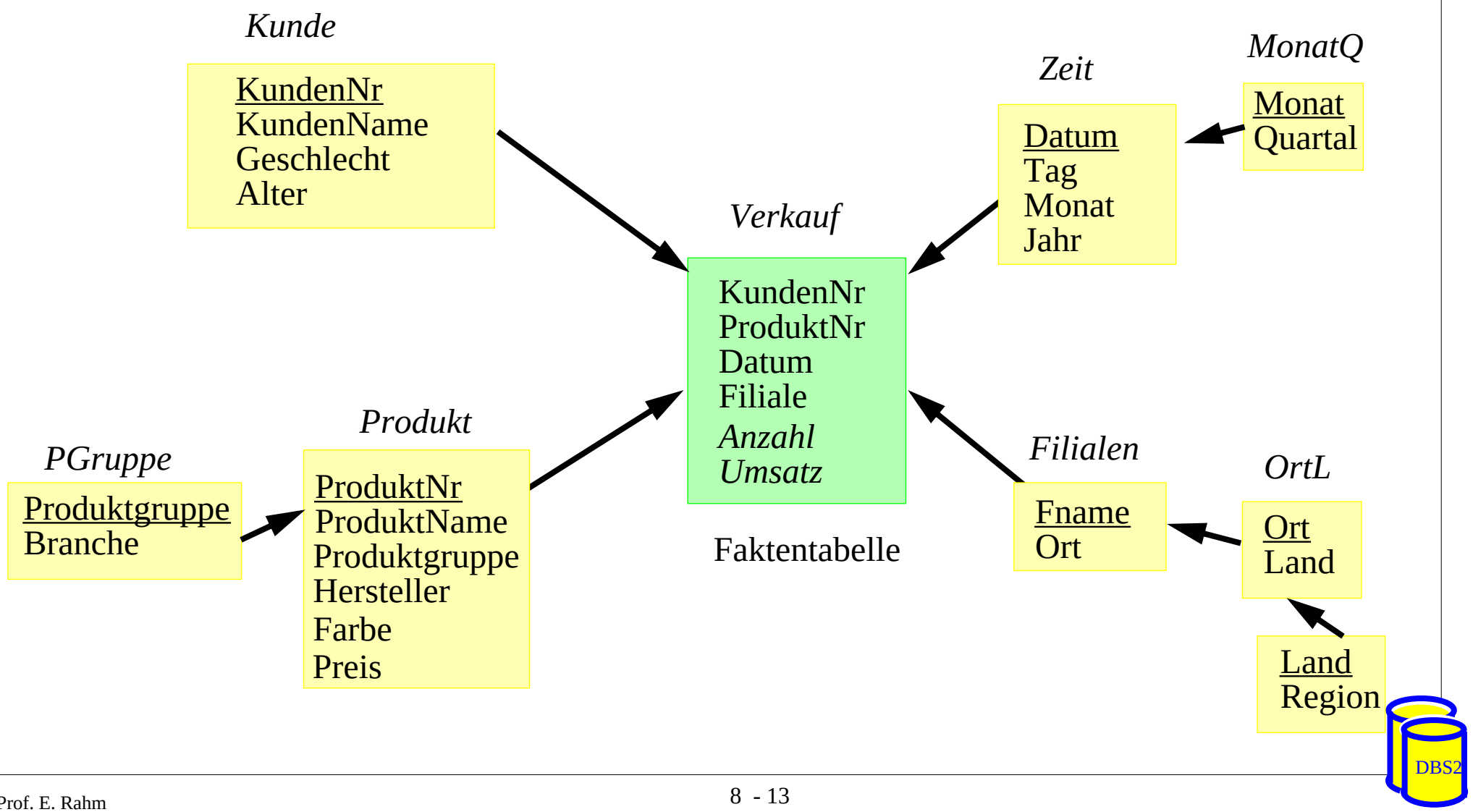
Star-Schema

- zentrale Faktentabelle sowie 1 Tabelle pro Dimension
- denormalisierte Dimensionstabellen



Snowflake-Schema

- explizite Repräsentation der Dimensionshierarchien
- normalisierte Dimensionstabellen



Beispielausprägung eines Data Warehouses

Verkauf					
Datum	Filiale	ProduktNr	KundenNr	Anzahl	Umsatz
7654 ...	Leipzig4 ...	1847 ...	4711 ...	1 ...	40549 ...

Filialen			
FName	Ort	Land	Region
Leipzig4 ...	Leipzig	Sachsen ...	Ost ...

Kunde			
KundenNr	Name	Geschlecht	Alter
4711 ...	Weber ...	M ...	39 ...

Zeit					
Datum	Tag	Monat	Jahr	Quartal	...
7654 ...	25 ...	Juni ...	1998 ...	2 ...	...

Produkt					
ProduktNr	Produktname	Produktgruppe	Hersteller	Farbe	Preis
1847 ...	Passat XY ...	Auto ...	VW ...	Blau ...	42999 ...

Anfragen

- Star-Join
  - sternförmiger Join der (relevanten) Dimensionstabellen mit der Faktentabelle
  - Einschränkung der Dimensionen
  - Verdichtung der Kennzahlen durch Gruppierung und Aggregation

Beispielanfrage: Welche Auto-Hersteller wurden von weiblichen Kunden in Sachsen im 1. Quartal 1998 favorisiert?

select p.Hersteller, **sum** (v.Anzahl)
from Verkauf v, Filialen f, Produkt p, Zeit z, Kunden k
where z.Jahr = 1998 **and** z.Quartal = 1 **and** k.Geschlecht = 'W' **and**
p.Produkttyp = 'Auto' **and** f.Land = 'Sachsen' **and**
v.Datum = z.Datum **and** v.ProduktNr = p.ProduktNr **and**
v.Filiale = f. FName **and** v.KundenNr = k.KundenNr
group by p.Hersteller;

Roll-Up/Drill-Down-Anfragen

- Roll-Up: Elimination eines Attributs aus der **group by**-Klausel
- Drill-Down: Hinzufügen eines Attributs in die **group by**-Klausel

```
select Jahr, Hersteller, sum (Anzahl)
from Verkauf v, Produkt p, Zeit z
where v.ProduktNr = p.ProduktNr and v.Datum= z.Datum  and p.Produkttyp = 'Auto'
group by p. Hersteller, z. Jahr;
```

```
select Jahr, sum (Anzahl)
from Verkauf v, Produkt p, Zeit z
where v. Produkt = p. ProduktNr and v.Datum = z.Datum   and p. Produkttyp = 'Auto'
group by z. Jahr;
```

```
select sum (Anzahl)
from Verkauf v, Produkt p
where v. Produkt = p. ProduktNr and p. Produkttyp = 'Auto';
```

Ergebnisse der Anfragen

Autoverkäufe nach Hersteller und Jahr		
Hersteller	Jahr	Anzahl
VW	1995	2.000
VW	1996	3.000
VW	1997	3.500
Opel	1995	1.000
Opel	1996	1.000
Opel	1997	1.500
BMW	1995	500
BMW	1996	1.000
BMW	1997	1.500
Ford	1995	1.000
Ford	1996	1.500
Ford	1997	2.000

Autoverkäufe nach Jahr	
Jahr	Anzahl
1995	4.500
1996	6.500
1997	8.500

Autoverkäufe nach Hersteller	
Hersteller	Anzahl
VW	8.500
Opel	3.500
BMW	3.000
Ford	4.500

Autoverkäufe
Anzahl
19.500

Kreuztabellen (Crosstab)-Darstellung

Hersteller / Jahr	1995	1996	1997	Σ
VW	2.000	3.000	3.500	8.500
Opel	1.000	1.000	1.500	3.500
BMW	500	1.000	1.500	3.000
Ford	1.000	1.500	2.000	4.500
Σ	4.500	6.500	8.500	19.500

Materialisierung von Aggregaten

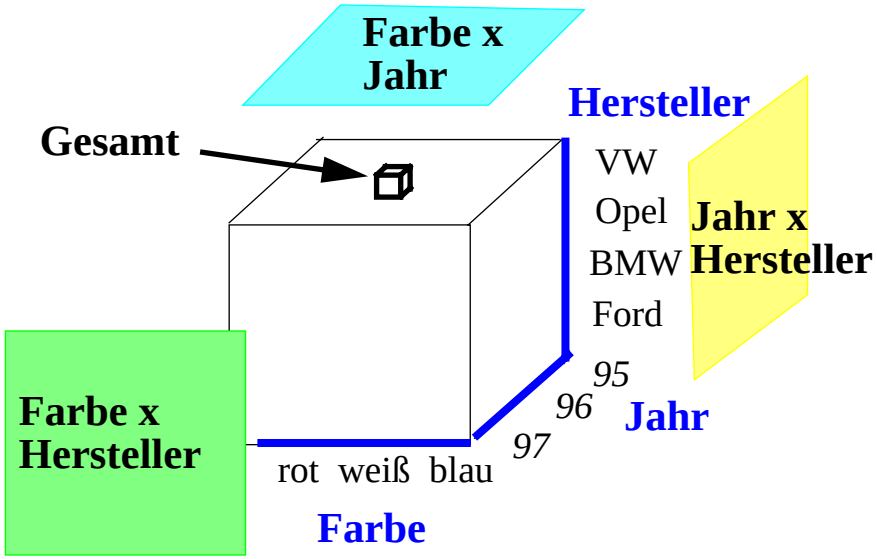
```
create table Auto2DCube (Hersteller varchar (20), Jahr integer, Anzahl integer);

insert into Auto2DCube
     (select p.Hersteller, z.Jahr, sum (v. Anzahl)
       from Verkauf v, Produkt p, Zeit z
       where v.ProduktNr = p.ProduktNr and p.Produkttyp = 'Auto'  and v.Datum = z.Datum
       group by z.Jahr, p.Hersteller)
  union
    (select p.Hersteller, to_number (null), sum (v.Anzahl)
     from Verkauf v, Produkt p
     where v.ProduktNr = p.ProduktNr and p.Produkttyp = 'Auto'
    group by p. Hersteller)
  union
    (select null, z. Jahr, sum (v.Anzahl)
     from Verkauf v, Produkt p, Zeit p
     where v.ProduktNr = p ProduktNr and p.Produkttyp = 'Auto' and v.Datum = z.Datum
     group by z. Jahr)
  union
     (select null, to_number (null), sum (v.Anzahl)
      from Verkauf v, Produkt p
      where v.ProduktNr = p. ProduktNr and p.Produkttyp = 'Auto');
```

Ausprägung der Relationen

Auto2DCube		
Hersteller	Jahr	Anzahl
VW	1995	2.000
VW	1996	3.000
VW	1997	3.500
Opel	1995	1.000
Opel	1996	1.000
Opel	1997	1.500
BMW	1995	500
BMW	1996	1.000
BMW	1997	1.500
Ford	1995	1.000
Ford	1996	1.500
Ford	1997	2.000
null	1995	4.500
null	1996	6.500
null	1997	8.500
VW	**null**	8.500
Opel	**null**	3.500
BMW	**null**	3.000
Ford	**null**	4.500
null	**null**	19.500

Auto3DCube			
Hersteller	Jahr	Farbe	Anzahl
VW	1995	rot	800
VW	1995	weiß	600
VW	1995	blau	600
VW	1996	rot	1.200
VW	1996	weiß	800
VW	1996	blau	1.000
VW	1997	rot	1.400
...	...	...	...
Opel	1995	rot	400
Opel	1995	weiß	300
Opel	1995	blau	300
...	...	...	...
BMW	...	...	...
...	...	...	...
null	1995	rot	...
null	1996	rot	...
...	...	...	...
VW	**null**	**null**	8.500
...	...	...	...
null	**null**	**null**	19.500

Cube-Operator



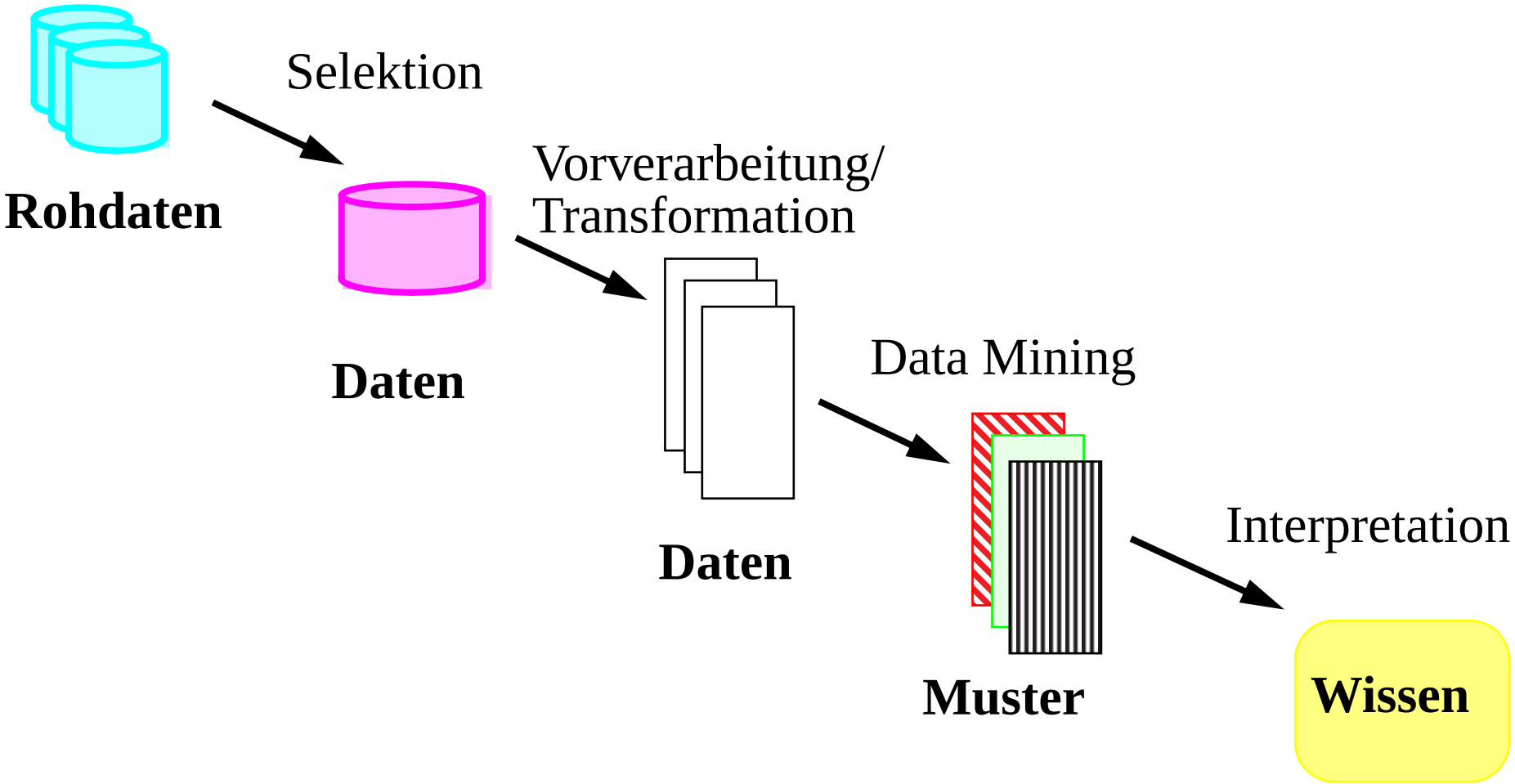
- Ausgangspunkt sind n-dimensionale Aggregate: a1, a2, ... , an, f ()
- Bildung von Super-Aggregaten über (n-1)-dimensionale Sub-Cubes
  ALL, a2, ..., an, f ()
  a1, ALL, ..., an, f ()
  . . .
  a1, a1, ... an, f ()
- Bildung von Super-Aggregaten über (n-2)-dimensionale Sub-Cubes
  ALL, ALL, a3 ..., an, f ()
  • • •
  a1, a2, ..., ALL, ALL, f ()

Der Cube-Operator (2)

select p. Hersteller, z. Jahr, p.Farbe, **sum** (v. Anzahl)
from Verkauf v, Produkt p, Zeit z
where v.ProduktNr = p. ProduktNr **and** p.Produkttyp = 'Auto' **and** v.Datum = z.Datum
group by z.Jahr, p.Hersteller, p.Farbe **with cube**;

- Man erspart sich die Formulierung von 2^n **union**-Anfragen (bei n Attributen in der **group by**-Klausel)
- Die Aggregation geht sehr viel effizienter, da Zwischenergebnisse vom DBS wiederverwendet werden können

Knowledge Discovery



Data Mining: Techniken

- Clusteranalyse
  - Objekte werden aufgrund von Ähnlichkeiten in Klassen eingeteilt
- Assoziationsregeln
  - Warenkorbanalyse (z.B. Kunde kauft A und B => Kunde kauft C)
- Entscheidungsbaum-Verfahren
  - Klassifikation von Objekten
  - Erstellung von Klassifikationsregeln (z.B. “guter Kunde” wenn Alter > 25 und ...)
- Neuronale Netze
  - Klassifikation
  - Erstellung von Klassifikationsregeln
  - Vorhersage von Attributwerten
- Genetische Algorithmen
  - multivariate Optimierungsprobleme (z.B. Identifikation der besten Bankkunden)

Zusammenfassung

- Data Warehousing: Anfrageverarbeitung auf separatem Datenbestand für Decision Support (OLAP)
- Hauptschwierigkeit: Integration heterogener Datenbestände sowie Bereinigung von Primärdaten
- sehr hohe Anforderungen an DB-Verarbeitung
  - riesige Datenvolumina
  - mehrdimensionale Auswertungen
  - temporale Anfragen
  - Notwendigkeit neuer Aggregate (Rank, Percentile, Moving Average, ...)
  - Skalierbarkeit
- Techniken zur Unterstützung einer hohen Leistungsfähigkeit
  - Vorausberechnung / Materialisierung von häufig benötigten Aggregaten
  - spezielle Indexstrukturen (z.B. Bit-Indizes)
  - parallele Anfrageverarbeitung
- Data Mining: Einsatz wissensbasierter Methoden auf Basis von Data Warehouses
  - Aufinden von Korrelationen, Mustern und Trends in Daten
  - setzt im Gegensatz zu Data Warehousing/OLAP (“knowledge verification”) kein formales Modell voraus (“knowledge discovery”)